******华中科技大学2023~2024第二学期**

**“ 计算机系统基础 ”考试试卷 (B卷)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **考试方式** | **闭卷** | **考试时间** | **2024年5月25日** | **考试时长** | **150 分钟** |

**院（系）：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 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） 数据的存储和访问**

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

struct person {

char id[6];

int height;

double salary;

char grade;

};

void f( )

{ int x=25; // &x 为 0xffffd4e0

int \*px = &x; // &px 为 0xffffd4e4

short sa[2]={-2, 5}; // &sa (即 &sa[0]) 为 0xffffd4e8

char msg[6]="abcd"; // &msg[0]为0xffffd506，'a' 的 ASCII为 0x61

struct person p; // &p 为 0xffffd4ec, &(p.salary)为 0xffffd4f8

// p中各字段的地址被对应类型长度整除

strcpy(p.id, "\*\*"); ① \*\* 应被替换为 \_\_**1234**\_\_\_\_\_\_\_\_

p.height = $$; ② $$ 应被替换为 \_\_**0x64**\_\_\_\_\_\_\_

p.grade='@@'; ③ @@ 应被替换为 \_\_ **b** \_\_\_\_\_\_\_\_

p.salary = 12000; ④ p.salary 的表示为 0x\_40 c7 70 00 00 00 00 00\_\_

⑤ person的大小为 \_\_24\_\_\_\_字节。

}

下面是以16进制数的形式显示内存内容，每空都需要填一个字节，最左边是内存地址。

注：数据是小端存储方式；部分内存单元内容与结构变量中的字段无关，具有随机性。

0xffffd4e0 \_19\_\_ \_00\_\_ \_00\_\_ \_00\_\_ \_e0\_\_ \_d4\_\_ \_ff\_\_ \_ff\_\_

0xffffd4e8 \_fe\_\_ \_ff\_\_ \_05\_\_ \_00\_\_ 31 32 33 34

0xffffd4f0 00 e4 fb f7 64 00 00 00

0xffffd4f8 00 00 00 00 00 70 c7 40

0xffffd500 62 d5 ff ff 6c e6 \_61\_\_ \_62\_\_

0xffffd508 \_63\_\_ \_64\_\_ \_00\_\_ XX XX XX XX XX

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

int global[5]; // 全局变量

void data\_access( )   
{

…………. // 此处有一些机器指令设置了 %esp，%eax 等

**int i =0 ;**

movl $0x0, (%esp)

**int local[5];**

**int \*p =NULL;**

movl $0x0, 0x4(%esp)

**global[0] = 1;**

movl $0x1, 0x34(%eax)

**global[ \_1\_\_ ] = \_\_2\_\_;**

movl $0x2, 0x38(%eax)

\_\_\_\_ **i = 2**\_\_\_\_\_\_\_;

movl $0x2, (%esp)

\_\_\_\_ **i++**\_\_\_\_\_\_\_;

addl $0x1, (%esp)

\_\_\_\_ **global [ i ] = 3**\_\_\_\_\_;

mov (%esp), %edx

movl $0x3, 0x34(%eax,%edx,4)

**\*(global +\_\_3\_\_）= 10 ;**

movl $0xa, 0x40(%eax)

**local[0] =1;**

movl $0x1, 0x8(%esp)

\_\_\_\_ **local [ 3 ] = 20**\_\_\_\_\_;

movl $0x14, 0x14(%esp)

\_\_\_\_\_ **p = &local [0]**\_\_\_\_\_;

lea 0x8(%esp), %eax

mov %eax, 0x4(%esp)

\_\_\_\_\_ **p += 4**\_\_\_\_\_\_;

addl $0x10, 0x4(%esp)

\_\_\_\_\_**\*p = 0x32**\_\_\_\_;

mov 0x4(%esp),%eax

movl $0x32, (%eax)

\_\_\_\_\_ **i = local [2]**\_\_\_\_;

mov 0x10(%esp),%eax

mov %eax, (%esp)

……………..   
}

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

**二、（20分） C语言常用运算及函数调用的机器级实现**

**1、写出各条C语句执行后相应变量的值，并将机器指令补充完整 （10分）。**

void myfunc( )

{

…………

**int x, y;**

**unsigned int u, v;**

**unsigned short s = 0xfffe;**

0x0000119d : movw $0xfffe,-0x14(%ebp)

**short t = -1;**

0x000011a3 : movw $0xffff,-0x12(%ebp)

**x = s ; // ① x = 0x\_0000fffe\_**

0x000011a9 : movzwl -0x14(%ebp), \_\_**%eax**\_\_

0x000011ad : mov %eax,-0x10(%ebp)

**y = t ; // ② y = 0x\_ffffffff\_**

0x000011b0 : \_ **movswl**\_\_ -0x12(%ebp),%eax

0x000011b4 : mov %eax,-0xc(%ebp)

**u = s; // ③ u = 0x\_0000fffe\_\_**

0x000011b7 : \_**movzwl**\_\_ -0x14(%ebp),%eax

0x000011bb : mov %eax,-0x8(%ebp)

**v = t; // ④ v = 0x\_ffffffff\_**

0x000011be : \_\_**movswl\_\_ -0x12(%ebp),%eax**\_\_\_

0x000011c2 : mov %eax,-0x4(%ebp)

**if (x > 0) t = 1; // ⑤ t = 0x\_\_1\_\_**

0x000011c5 : cmpl $0x0,-0x10(%ebp)

0x000011c9 : **\_jle\_\_** 0x11d1 <myfunc+68>

0x000011cb : movw $0x1,-0x12(%ebp)

**if (u > 0) t = 2; // ⑥ t = 0x\_\_2\_\_\_**

0x000011d1 <+68>: cmpl $0x0,-0x8(%ebp)

0x000011d5 : **\_jbe\_\_** 0x11dd <myfunc+80>

0x000011d7 : movw $0x2,-0x12(%ebp)

**u = u >> 2; // ⑦ u = 0x\_3fff\_\_\_**

0x000011dd <+80>: **\_shrl $0x2\_\_**, -0x8(%ebp)

**x = x >> 2; // ⑧ x = 0x\_3fff\_\_\_**

0x000011e1 <+84>: **\_sarl $0x2\_\_\_,** -0x10(%ebp)

**y = -y; // ⑨ y = 0x \_\_1\_\_**

0x000011e5 <+88>: **\_negl\_\_** -0xc(%ebp)

**v = ~v; // ⑩ 按位取反; v = 0x\_\_0\_\_**

0x000011e8 <+91>: **\_notl\_\_** -0x4(%ebp)

…………

}

1. 函数调用的机器级实现 （10分）

语句“int r = fsub(25,10);”对应的反汇编代码（最左边的是机器指令的地址）如下。

|  |  |
| --- | --- |
| **0x0f** | 0xffffd4e4 |
| **0xffffd508** | 0xffffd4e8 |
| **0x565561c4** | 0xffffd4ec |
| **0x19** | 0xffffd4f0 x |
| **0xa** | 0xffffd4f4 y |
| XXXXXXXX | 0xffffd4f8 |

0x565561bb : 6a 0a push $0xa

0x565561bd : 6a 19 push $0x19

0x565561bf : e8 c9 ff ff ff call 0x5655618d <fsub>

0x565561c4 : 83 c4 08 add $0x8,%esp

0x565561c7 : 89 45 fc mov %eax,-0x4(%ebp)

设执行“int r = fsub(25,10);”之前，esp的值为0xffffd4f8， ebp 值为 0xffffd508。

函数fsub的反汇编代码如下：

int fsub(int x, int y) {

0x5655618d <+0>: 55 push %ebp

0x5655618e <+1>: 89 e5 mov %esp,%ebp

0x56556190 <+3>: 83 ec 10 sub $0x10,%esp

0x56556193 <+6>: e8 35 00 00 00 call 0x565561cd <\_\_x86.get\_pc\_thunk.ax>

0x56556198 <+11>: 05 44 2e 00 00 add $0x2e44,%eax

int result;

result = x- y;

0x5655619d <+16>: 8b 45 08 mov 0x8(%ebp),%eax

0x565561a0 <+19>: 2b 45 0c sub 0xc(%ebp),%eax

0x565561a3 <+22>: 89 45 fc mov %eax,-0x4(%ebp)

return result;

0x565561a6 <+25>: 8b 45 fc mov -0x4(%ebp),%eax

}

0x565561a9 <+28>: c9 leave

0x565561aa <+29>: c3 ret

在上面的表格中，填写执行到 leave 时相应内存单元中的值（表格中的5行都需要填写，每行均用0x作为前缀）。

① 函数参数 x的地址（即&x）是 0x\_\_\_ **ffffd4f0**\_\_\_\_\_\_\_\_\_\_。

② 局部变量 result的地址（即&result）是 0x\_\_ **ffffd4e4**\_\_\_\_\_\_\_\_\_\_\_。

③ 执行 “sub 0xc(%ebp), %eax” 后， CF=\_**0**\_\_\_, SF=\_\_**0**\_\_\_, ZF=\_\_**0**\_\_\_, OF= \_\_**0**\_\_\_。

④ 执行 “ret ” 后 ，%esp = 0x \_\_**0xffffd4f0**\_\_\_\_\_\_\_\_\_\_\_

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

**三、（20分）计算机系统的基本原理**

1、程序计数器（eip）的自动变化（10分）

设有 void myfunc( ) { ……}

int main( ) { ……

**int x = -20;**

0x565561ca <+29>: c7 45 f0 ec ff ff ff movl $0xffffffec,-0x10(%ebp)

// 在取出该指令并进行译码后，eip = 0x\_\_**565561d1**\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**int y;**

**void (\*fp)();**

**if (x >= 0) y = 1;**

0x565561d1 <+36>: 83 7d f0 00 cmpl $0x0,-0x10(%ebp)

0x565561d5 <+40>: 78 09 js 0x565561e0 <main+51>  
 // 上面语句中的 0x565561e0 ，是\_\_**(eip) + 09**\_\_ 计算得到的

// 当js转移条件成立，即sf =\_\_**1**\_\_ 时，执行 (eip) + 指令中位移量 → eip

// 若js转移条件不成立，该指令也就执行完成，此时 eip = 0x\_**565561d7**\_\_

0x565561d7 <+42>: c7 45 ec 01 00 00 00 movl $0x1,-0x14(%ebp)

0x565561de <+49>: eb 07 jmp 0x565561e7 <main+58>

// 上面语句中的机器码 07 ，表示的含义是\_**目标地址与当前eip 之前的位移量**\_\_

**else y = -1;**

0x565561e0 <+51>: c7 45 ec ff ff ff ff movl $0xffffffff,-0x14(%ebp)

**myfunc();**

0x565561e7 <+58>: e8 b1 ff ff ff call 0x5655619d <myfunc>

// 执行call 指令时，会将\_\_**将 eip 压栈**\_\_\_\_， 将\_\_**0x5655619d**\_\_→ eip

**fp = myfunc;**

0x565561ec <+63>: 8d 83 c5 d1 ff ff lea -0x2e3b(%ebx),%eax

// 执行 lea 指令时，ebx = 0x \_**56558fd8 或0x5655619d + 0x2e3b** \_\_\_

0x565561f2 <+69>: 89 45 f4 mov %eax,-0xc(%ebp)

**fp();**

0x565561f5 <+72>: 8b 45 f4 mov -0xc(%ebp),%eax

0x565561f8 <+75>: ff d0 call \*%eax

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

0x565561fa <+77>: . ………

}

执行函数myfunc中的ret 指令时，CPU会 从栈顶弹出一个双字 →eip。

**2、程序执行的基本原理（10分）。**

**（1）程序运行的基本过程是什么？**

**A：取指令，cpu根据eip的值从内存中读入指令； B、指令译码， eip自增指令的长度； C、取操作数，计算操作数的地址并根据地址取得操作数； D、执行，在算术逻辑运算部件执行相应的操作； E、结果写回。重复上述步骤，使得程序自动地一条一条执行。**

**（2）举例说明什么是地址类型转换？什么是数据类型转换？**

**地址类型转换：int v; char \*p = (char \*)&v; 相同的内存单元，视为存放不同类型的数据。 数据类型转换：int v=1; float f= (float)v; 相同的值1，整型、浮点型有不同的表现形式。**

**（3）全局变量与非静态的局部变量各分配在什么段中？ 为什么一个函数不应返回局部变量的地址？**

**全局变量在数据段中分配空间，非静态局部变量在堆栈段中分配空间。**

**返回（非静态）局部变量的地址，也就是返回栈中一个单元的地址。堆栈是各个函数共用的空间；函数在返回后，函数之前占用的堆栈空间被其他函数所用，根据该地址进行后续操作时，访问到的数据可能并非原来函数结束前的值，存在不确定性。**

**（4）****指令集体系结构与计算机系统的什么部件有关？它是哪两个部分之间的接口？**

**指令集体系结构（ISA）与CPU 有关，CPU决定了 ISA。**

**ISA是 软件和硬件之间的接口。各种程序都是由指令构成， 而指令交给CPU去执行。**

**（5）什么是缓冲区溢出？缓冲区溢出会有哪些危害？**

**缓冲区溢出：读/写缓冲区时，超出了缓冲区定义的范围。**

**缓冲区溢出可能会造成程序出现逻辑错误（如非预期修改了其他变量的值）、异常、甚至崩溃（如越出本程序的空间分配范围）。**

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

**四、（20分）程序编译及优化**

1、举例说明编译器利用CPU的特性所能进行的优化。请给出5种不同类型的示例，并说明优化原理。（10分）

1. 利用指令流水线：循环展开、分支语句向无分支转换等等。当遇到转移时，指令流水线上所做的一些工作将被抛弃，流水线的效率受到影响。
2. 利用cache：尽量按顺序访问内存单元（例如将按列访问二维数组调整为按行访问）， 否则会产生 cache抖动，即一个数据块的频繁换入换出cache，增加时间开销。
3. 利用更快的机器指令：例如lea -0x7(%eax, %eax, 8), %eax 实现 9\*(eax) – 7 => eax。不同指令的执行速度不同。
4. 利用串操作指令：代替对串中的数据一个一个循环操作（如将一个存储区中内容拷贝到另一个存储区中）。
5. 使用SIMD：一条指令可以同时操作多个操作数，对于批量数据减少操作次数，节约时间。
6. 寄存器与变量绑定：不需要访问内存单元（变量）。
7. 使用多核CPU：多线程编程。

**2、除了利用CPU特性进行优化，编译器还可以优化减少在CPU上执行指令。举出三种不同类型的优化示例。（6分）**

(1) 去除“废”语句（废语句段）：

int x = 1; for(x = 2; x < 10; x++) { … } =>

int x; for(x = 2; x < 10; x++) { … } 有这些语句与没有这些语句的运行结果是等同。

(2) 可以计算出的结果：

int x = 1; int y = x + 2; int z = f(x); int f(int x) { return x + 1; } =>

int x = 1; int y = 3; int z = 2;

(3) 将函数调用转换成直接使用函数体的形式：

减少函数参数的传递，减少CALL与 RET指令。

(4) 减少部分运算

int f(int x,int y) {int u=2\*x; int v=3\*y+5; return u+v; }   
当函数调用为 w=f(10,a); 可以省掉 int u=2\*x, 直接使用 return 25+3\*y;

**3、为了提高程序的安全性，编译器可以在程序中生成一些安全检查代码。举出两种不同的安全检查示例并说明其原理。（4分）**

堆栈帧检查：访问存储空间是否发生溢出，例如 int a[2]; a[2]=10; 发生了溢出。

为数组a 的前后，各预留一个4字节的空间，初值置为0xcccccccc; 在程序结束前，检查预留的空间中的内容是否还是 0xcccccccc；若不是，则说明发生了溢出。

未初始化变量使用检查：检查一个变量在使用之前是未初始化。

示例：int x; if （flag==1）x=10; int y=x ; 注：编译器可以肯定未初始化就使用的变量，在编译时就会报错；对于有条件初始化变量（如flag==1时），则编译器可能无法判定是否初始化。  
 对于有条件初始化变量x，编译器为其生成一个影子变量，初值为0。对于 x的赋值语句，同时生成为影子变量置1的指令；对与使用x的语句，则判断影子变量是否为1，确定是否使用了未初始化的变量。

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

**五、（20分） 链接和异常控制流**

1. 设有如下程序（10分）

**#include <stdio.h>**

**int global =20;**

**int main()**

**{ int x , y;**

**x = 25;**

**global = 25;**

**y = global;**

**printf("good %d %d\n",x, y);**

**return 0;**

**}**

在编译生成可重定位目标文件中，会有数据节（.data）、只读数据节（.rodata）、代码节（.text）、代码节的重定位节（.rel.text）、符号表节（.symtab）、字符串节（.strtab）等。

请结合给出的程序，回答如下问题。

（1）上述六个节中分别包含哪些信息？（6分）

.data： 为global分配4个字节，被设置为0x00000014

.rodata： "good %d %d\n"，0

.text： main()函数的机器指令

.rel.text：对于每一个需要中定位的条目，都会有如下重定位信息， 什么位置需要重定位、由哪一个符号来重定位、用什么方式来重定位，以及重定位置计算的附加置。

.symtab：所有符号（global、main、printf）的相关信息，如符号所在的节号、在节（.data）节的地址、所占空间的大小，等等。

.strtab： 所有符号的名字，如"global"、"main"、"printf"

（2）需要重定位的符号有哪几个？（2分）

3个符号：**global、main、printf**

（3） 哪些C语句和语句中的哪些部分需要重定位？（不用给出具体的机器指令）？（2分）

**global = 25； global**

**y = global； global**

**printf("good %d %d\n", x, y)； printf 和 "good %d %d\n" 都需要重定位**

**2、异常控制流** （10分）

（1） 产生中断的原因有哪些？

不可屏蔽事件（如掉电）、可以屏蔽的外部事件（如敲击键盘、鼠标、时钟中断等）。

（2） 产生异常的原因有哪些？

CPU 执行指令时引起的异常事件（如软中断、除法出错、单步中断、缺页、访问单元地址越界等）。

（3）中断和异常的差别什么？

中断是由外部设备触发、与正在执行的指令无关的异步事件。异常是与正在执行的指令相关的同步事件（与外部设备无关）。

（4）什么是中断描述符表？

存放中断处理程序相关信息（入口地址、类别、权限等）的表。

（5） 中断和异常的响应过程是什么？

CPU捕捉到中断事件，CPU暂停正在进行的程序，根据中断描述符表获取中断服务程序的入口地址，然后执行中断服务程序，执行完后返回到程序暂停的位置继续执行。异常的响应过程是类似的，异常处理后，可以重新执行引起异常的指令（故障），或者引起异常指令的下一条指令（陷阱），或者终止。