******华中科技大学计算机科学与技术学院2022~2023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** | **2022-12-04** | **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 | 20 | 10 | 100 |  |
| **得分** |  |  |  |  |  |  |  |

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

1. **计算机工作基本原理填空（共20分，每空1分）**

1、计算机采用存储程序的工作方式，自动逐条取出和执行指令。指令在内存中的存储地址由程序计数器指明（Intel CPU中为指令指示器EIP 或者RIP，下面以 EIP为例）。EIP自动变化的规则和变化方式如下：

① 在取出指令、对指令进行译码后，EIP会自动加上\_\_\_当前指令的字节长度 \_\_\_\_\_\_\_\_；若该指令不涉及转移，则可以实现程序的顺序执行；

② 若指令是条件转移指令，如\_\_\_ JE \_\_\_LP (LP为一个标号)，在ZF=1时，会执行（EIP）+位移量→ EIP , 其实质操作是将\_\_ LP 的偏移量 送到EIP中；

③ 若指令是无条件转移指令JMP， 除了直接跳转外，还可以进行间接跳转。借助于寄存器EAX，通过间接跳转实现与“JMP LP”相同的功能，可以使用语句：

JMP EAX

④ 借助堆栈段和子程序返回指令，实现与“JMP LP”相同的功能，可以使用语句：

PUSH LP

RET

⑤ 执行 CALL 指令时，CPU 会\_\_\_ 将返回地址（EIP 的下一条指令地址）压栈，并跳转到目标地址 \_\_；

⑥在程序运行过程中，出现中断信号时，CPU会根据中断类型号，利用\_\_\_ IDT \_\_\_\_\_\_\_\_\_找到中断处理程序的入口地址。中断处理程序中有\_\_\_ IRET \_\_指令，从栈中取出进入中断处理程序前保存的断点地址送给 EIP，从而实现被中断程序的继续执行。

2、机器指令一般都有操作对象，指令中可以直接存放常量操作数或者操作数的地址。操作数的地址有多种表达方式。 下面一段代码给出了访问同一单元的多种C 语句。请回答C语句访问方式与机器指令的对应关系。

int global[5] = { 10,20,30,40,50 }; // 全局变量

void func ( )

{ int local[5] = { 10,20,30,40,50 };

int i = 2;

int \*p;

global[i] = 25;

\*(global + 2) = 25;

p = &global[2];

\*p = 25;

local[i] = 25;

}

① 在反汇编窗口看到global[i]=25对应如下两条指令：

mov eax, dword ptr [ebp-18h] ; 显示符号名时的语句为 mov eax, dword ptr [i]

mov dword ptr [eax\*4+00419000h],19h

第一条指令中变量i的地址表达式为 \_\_\_\_ \_\_\_\_\_\_\_\_\_ ；第二条指令中目的操作数的寻址方式是\_\_\_\_\_\_ \_\_\_\_\_\_\_，数组global的起始地址是 \_\_\_ \_\_\_\_\_\_，目的操作数寻址中采用比例因子4的原因是\_\_ \_\_\_\_\_\_。

② \*(global + 2) = 25; 仅对应一条机器指令，指令为 \_\_ \_\_\_\_

其目的操作数的寻址方式是 \_\_\_\_\_ \_\_\_\_\_\_\_\_ 。

③ \*p = 25; 如果变量p 的地址表达式为 [ebp-1Ch]，访问存储单元(\*p)时，使用的是寄存器间接寻址方式（使用寄存器 eax），则对应的两条汇编指令为：

\_\_ \_

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

④ local[i]=25；访问local[i] 采用的是基址加变址寻址方式，对应的汇编指令如下：

mov \_\_ \_\_\_\_\_\_, dword ptr [ebp-18h]

mov dword ptr [ebp+eax\*4-14h],19h

与全局变量global空间分配位置不同，local的空间分配在栈上。

数组local的起始地址的表达式为：\_\_\_\_\_ \_\_\_\_\_\_\_\_。

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

1. **数据存储及C语句转换填空（共30分）**

在Windows + VS2019（Intel CPU）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 fmin(int x, int y)

{

int t;

if (x > y) t= y;

else t = x;

return t;

}

void fmain( )

{

char buf[6] = "12345"; // '1'的ASCII是 0x31

int u = 0x20221204; // mov dword ptr [ebp-0Ch], 20221204h

int v = -32; // mov dword ptr [ebp-10h], 0FFFFFFE0h

int w = 0; // mov dword ptr [ebp-14h],0

w = fmin(u, v);

w = \*(int \*)(buf+1);

}

在调试时，在监视窗口中看到数组 buf 的起始地址为 0x00dffdb8，变量 u 的地址（即&u）为 0x00dffdb4； v 的地址(即&v) 为 0x00dffdb0。

1. 填写执行到“w=fmin(u, v)；”时，内存中数据存放的结果。要求以字节为单位、用16进制数的形式填空，最左边是内存窗口显示的内存地址。（6分）
2. 0x00dffdb0 \_\_ \_ \_\_ \_ \_\_ \_ \_\_ \_ \_\_ \_ \_\_ \_ \_\_ \_ \_\_ \_
3. 0x00dffdb8 \_\_ \_ \_\_ \_ \_ \_\_ \_\_ \_ \_ \_\_ \_\_ \_\_ XX XX
4. 数据传送指令解读 （6分，每空 2分）

语句 “int u = 0x20221204;”的反汇编指令为： **m**ov dword ptr [ebp-0Ch], 20221204h

根据观察到的 u 的地址， 推断执行该语句时， (ebp)= 0x \_ \_\_\_\_, 该指令的机器码为 C7 45 F4 04 12 22 20，其中 F4 对应的是指令中\_\_\_ \_\_\_\_\_\_\_\_的编码。

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

1. 函数调用语句解读 （10分，每空2分）

对于语句“w=fmin(u, v);”对应的反汇编代码（最左边的是机器指令的地址）如下。

|  |  |
| --- | --- |
|  | 0x00dffd4c |
|  | 0x00dffd50 |
|  | 0x00dffd54 |
|  | 0x00dffd58 |
|  | 0x00dffd5c |
| XXXXXXXX | 0x00dffd60 |

009A204B mov eax,dword ptr [ebp-10h]

009A204E push eax

009A204F mov ecx,dword ptr [ebp-0Ch]

009A2052 push ecx

009A2053 call 009A125D

009A2058 add esp,8

009A205B mov dword ptr [ebp-14h],eax

设执行“w=fmin(u ,v);”之前，(esp) =0x00dffd60

① 在表格的适当位置填写 刚进入函数fmin内部时，堆栈中存放的相关数据。

② 刚进入函数fmin内部时，(esp)= 0x\_\_\_\_ \_\_\_\_\_\_\_

③ 执行 “ add esp, 8 ” 之后，(esp) = 0x\_\_\_\_ \_\_\_\_\_\_\_。

4. 函数fmin的指令解读 (8分，每小题2分)

函数体对应的反汇编代码有：

push ebp

mov ebp, esp

…… 此处有一些堆栈操作 push 指令，但无改变 ebp的指令

if (x > y) t = y;

009A20A3 mov eax,dword ptr [ebp+8]

009A20A6 cmp eax,dword ptr [ebp+0Ch]

009A20A9 jle 009A20B3

009A20AB mov eax,dword ptr [ebp+0Ch]

009A20AE mov dword ptr [ebp-4],eax

009A20B1 jmp 009A20B9

else t = x;

009A20B3 …………

009A20B9 …………

① 函数参数 x的地址（即&x）是 0x\_\_ \_\_\_\_。

② 局部变量 t的地址（即&t）是 0x\_ \_\_\_\_\_。

③ 执行 “cmp eax, dword ptr [ebp+0Ch] ” 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）。**

.686P

.model flat, c

includelib libcmt.lib

includelib legacy\_stdio\_definitions.lib

printf proto :ptr sbyte, :vararg

ExitProcess proto stdcall :dword

.data

buf sdword -100, 2022, -32, 90,-30

len = ($-buf)/4

fmt db "%d ", 0dh,0ah, 0

.code

main proc c

mov eax, 0

mov ecx, len ; ①

mov edi, offset buf ; ②

LP\_START:

cmp dword ptr [edi], 0

jle LP\_NEXT ; ③

inc eax

LP\_NEXT:

add edi, 4

sub ecx, 1

jne LP\_START

invoke printf, offset fmt, eax

invoke ExitProcess, 0

main endp

end

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buf数组中正数的个数到eax中，然后显示正数的个数。运行后显示2。

1. 若将 ③ 处的语句改为 “jbe LP\_NEXT”,程序运行的结果是什么？(2分)

显示5

1. 若标号 LP\_START 写到 ②处语句前，程序运行的结果是什么？为什么？（3分）

显示0．循环中始终比较buf的开头元素（0号元素）是否小于等于0，该值小于0.

1. 若标号 ①处语句写到标号LP\_START之后，程序运行会出现什么现象？为什么？（3分）

LP\_START: mov ecx, len cmp dword ptr [edi], 0 ……

**程序运行异常。表面上是死循环，每执行一次循环，ecx 都恢复成了 len。但是，由于循环中,edi 在不断地增加，到edi 指向的地址超出程序空间范围时，执行“cmp dword ptr [edi], 0”会出现访问异常，引起程序异常终止。**

**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小写字母转换为对应的大写字母，并将转换后的字符串、以及修改的字母个数输出。

……

.data

fmt db "%s %d" , 0

buf db "Hello, 123, Good", \_0\_\_\_

.code

main proc c

mov esi, \_\_\_\_0\_\_\_\_\_\_\_\_ ; esi 存放buf 数组元素的下标

mov ecx, \_\_\_0\_\_\_\_\_\_\_\_ ; ecx 存放修改字母的个数

start:

mov al, buf[esi]

cmp al, 0

je over\_\_\_\_/ jz over\_ \_\_\_

cmp al, 'a'

jl next\_\_\_\_/ jb next\_\_\_\_\_\_

cmp al, 'z'

jg next\_\_\_\_/ ja next\_\_\_\_\_\_

add al, \_\_\_’A’ – ‘a’\_\_

mov \_\_ buf[esi]\_\_\_ , al

inc ecx\_\_\_\_\_\_\_\_\_\_

next:

inc esi\_\_\_\_\_\_\_\_\_\_\_

jmp start

over:

invoke printf, offset fmt, offset buf, ecx

invoke ExitProcess, 0

main endp

end

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

1. **程序优化 （共20分）。**
2. 对C程序进行编译时，编译器可以做优化工作。请举出5个不同的优化场景，说明做了什么优化，以及能加快执行速度的原因。 (10 分)

① 访问二维数组时，将列序优先调整为行序优先。二维数组是按行序存放的，在访问数据时，数据会成块地调入 CPU的cache中，行序访问能够提高cache的命中率，避免不停地在内存和cache之间交换数据，从而提高执行速度。

② 用循环访问一维数组时，将循环展开，即变成无循环的语句，或者循环次数少的语句。因为CPU中采用指令流水线的方式，转移指令会导致流水线上的一些指令加工工作被废弃。若是顺序执行指令，则会充分发挥指令流水线的作用，提高速度。

③ 分支语句向无分支语句转换，例如使用条件传送指令代替转移指令。原理同②。

④ 用执行快的指令代替执行慢的指令。例如 x\*2， 用 x<<1 来代替。

CPU中的串操作指令、SIMD 指令，都可以用来加快程序的执行速度。

⑤ 执行算法的优化， 例如 3\*x, 可以变成 2\*x +x；而2\*x又可以使用逻辑左移1位指令来实现。虽然，指令条数变多，但速度会更快；原理就是 CPU执行指令的速度不一样，一条慢的指令比几条快的指令耗时更多。

上述优化是与计算机硬件设备相关的。其他优化包括软件层面的优化，如编译器就可以计算出值的表达式，可以直接得到结果，而不需要生成相应的机器指令序列。

2、如下的 C语言程序段的功能，其编译后调试版本的汇编语言代码如下(注：在反汇编时，勾选显示符号名)。(10分)

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

00862129 mov dword ptr [i],0

00862130 jmp fopt+5Bh (086213Bh)

00862132 mov eax,dword ptr [i]

00862135 add eax,1

00862138 mov dword ptr [i],eax

0086213B cmp dword ptr [i],5

0086213F jge fopt+70h (0862150h)

00862141 mov eax,dword ptr [i]

00862144 mov ecx,dword ptr [sum]

00862147 add ecx,dword ptr array[eax\*4]

0086214B mov dword ptr [sum],ecx

0086214E jmp fopt+52h (0862132h)

(1) 编写相应的优化汇编语言程序段，以提高程序的执行效率。(6分)

要求优化时保留循环结构，在优化程序段中可以用标号来代替转移指令的目的地址，要求写出寄存器的分配（即与变量的绑定关系）。

Ebx 存放变量 I 的值； eax存放变量 sum 的值 （1分）

mov eax, sum （sum的值赋给对应的寄存器 1分）

mov ebx, 0

loop\_p:

cmp ebx, 5 循环控制 1分；加法 1分

jge loop\_over 寄存器修改 1分

add eax, dword ptr array[ebx\*4]

inc ebx

jmp loop\_p

loop\_over:

mov dword ptr [sum], eax （寄存器的值送给sum 1分）

**mov i, ebx**

**(2)** 用循环展开的方法（即去除循环）优化程序段（4分）。

eax存放变量 sum 的值

mov eax, sum sum 与寄存器交换数据正确 1分

累加求和正确 2分； 最简表达形式 1分

add eax, array[0]

add eax, array[4]

add eax,array[8]

add eax, array[12]

add eax, array[16]

mov sum, eax

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

1. **程序的链接问答（10分）**
2. 在对一个C程序文件进行编译和汇编后，生成的可重定位目标文件(obj文件，或者 .o文件)，一般由哪些节组成（至少说出6个组成部分的名字）？（3分）

文件头、节头表、代码节.text, 已初始化数据节.data, 未初始化数据节.bss,

只读数据节 .rodata，符号表节 .symtab, 字符串表节 .strtab、代码节的重定位节(.rela.text)、数据节的重定位节(.rela.data) 等等。

1. C程序中，什么符号需要重定位？什么符号不需要重定位？（2分）

函数参数、非静态的局部变量不需要重定位；静态的局部变量、全局变量、函数需要重定位。

1. 设有一个函数中，有语句 int x=g; 其中g是一个初值为20的int 类型全局变量。编译器对g的定义(int g=20;)和x=g编译时，在可重定位目标文件中会生成哪些信息？（5分）

对于 int g =20; 在字符串节中，要存放变量的名字(ASCII串)；在符号表节要存放与g相关的信息（如g的值占存储空间的大小<4>；定义g的节<数据节>；g的类型<变量>；绑定方式<global>等等）；在数据节要存放 g的初值20。

对于 int x=g；在代码节要生成相应的指令，其中 对于g 的引用（即g的地址），先使用占位符（00 00 00 00）来填充。在代码节的重定位节，要产生一条相应的信息，表明代码节的什么位置要被替换成一个什么类型的值（重定位方式），该值来源于哪一个变量（符号表的哪一项）等。