四川大学期末考试试题（闭卷）

# （2018--2019 学年第 1 学期） A卷

课程号： **311046040** 课程名称：系统级编程任课教师：适用专业年级：软件工程 **2016**级学号：姓名：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 考生承诺  我已认真阅读并知晓《四川大学考场规则》和《四川大学本科学生考试违纪作弊处分规定（修订）》，郑重承诺：   1. 已按要求将考试禁止携带的文具用品或与考试有关的物品放置在指定地点； 2. 不带手机进入考场； 3. 考试期间遵守以上两项规定，若有违规行为，同意按照有关条款接受处理。   考生签名： | | | | | | | |
| 题号 | 一**(40%)** | 二**(10%)** | 三**(5%)** | 四**(15%)** | 五**(12%)** | 六**(8%)** | 七**(10%)** |
| 得分 |  |  |  |  |  |  |  |
| 卷面总分 |  | | 阅卷时间 |  | | | |

注意事项：1. 请务必将本人所在学院、姓名、学号、任课教师姓名等信息准确填写在试题纸和添卷纸上；

1. 请将答案全部填写在本试题纸上；
2. 考试结束，请将试题纸、添卷纸和草稿纸一并交给监考老师。



|  |  |
| --- | --- |
| 评阅教师 | 得分 |
|  |  |

## 一、单项选择题（本大题共20 小题，每小题2 分，共40 分）

提示：在每小题列出的四个备选项中只有一个是符合题目要求的，请将其代码填写在下表中。错选、多选或未选均无分。

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|  |  |  |  |  |  |  |  |  |  |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|  |  |  |  |  |  |  |  |  |  |

1. Which one can translate assembly into machine code?
   1. Debugger
   2. Assembler
   3. L oader
   4. Linker

答案：b

解析：

预处理器：将.c 文件转化成 .i文件。

编译器：将.c/.h文件转换成.s文件，也就是汇编代码。

汇编器：将.s 文件转化成 .o文件，也就是将汇编代码转换为机器代码。

链接器：将.o文件转化成可执行程序，将一个或多个目标文件转换为可执行文件。

加载器：将可执行文件加载到内存中并执行。

调试器：上述除加载器外全部都参与，将C变为可调试的机器代码，当开始调试时，加载器也参与。

1. Which expression is false?
   1. !(0)
   2. ~0xFF
   3. ~0x00
   4. 0x00^0x00

答案：b

解析：

A选项，在C中，0是false，非0是true，！是逻辑上的非，故其值为1，为true

B选项，位取反操作，FF全为1，故取反后为0，所以，其值为false

C选项，同样的位取反操作，全为1，故为true

D选项，异或操作，相同为0，不同为1，故其结果是非0数，所以为true

1. Given the following code segments, which statement can NOT alter ‘b’ in str1 to ‘d’?

char str1[]=“abc”; char \*p; a. str[1] = ‘d’;

* 1. \*(p++) = ‘d’;
  2. \*(str1++) = ‘d’;
  3. p=str1; p[1]=’d’;

答案：b

解析：

A选项，通过数组的下标直接修改的值。

B选项，p目前未赋值，是一个野指针，指向的地方不清楚。

C选项，数组名直接用表示数组的0号元素的地址，++，就指向1号元素。

D选项，先令p指向了数组首地址，简单的理解，现在的p就是str1，所以能修改。

1. Which statement has NO effect on stack?
   1. MOV
   2. POP
   3. PUSH
   4. CALL

答案：a

B、C选项就不说了，栈的压入和弹出，肯定有影响。

D选项，call是调用函数的指令，会改变栈帧的结构。

A选项，mov只能说最不对，是一个数据传输的指令。

1. The Visual C++ Memory window displays
   1. only local variables allocated in stack, without the associated variable names
   2. both local and global variables allocated in stack, without the associated variable names
   3. the contents of memory, interpreted in one of several ways, without the associated variable names
   4. the contents of memory, interpreted as 32-bit integers, without the associated variable names

答案：c，打开vc 6.0的内存窗口，显示的是地址，然后对内容的不同的解析方式，并没有显示出变量的名字。

1. What happens in a C program when an addition would cause integer overflow?
   1. The correct value is coerced to a floating point number.
   2. Execution is terminated.
   3. An incorrect result is produced and execution continues.
   4. An exception-handler is called with the two operands as parameters.

答案：c，可以自己去试一下，整数的溢出会导致不正确的结果，但不会使程序终止，C语言也不会报错。

1. The C expression a->b is equivalent to
   1. (&a) + b
   2. (\*a).b
   3. (&a).b
   4. \*(a + b)

答案：b，记住就行，这是C结构体的固定用法。

1. To quickly allocate and free many variables of a commonly used data type, we could
   1. use sizes which are powers of two.
   2. keep a linked list of free objects of that type's size.
   3. minimize the size of the data type.
   4. coalesce blocks when they are freed.

答案：b，做过的原题，快速分配和释放内存的一种数据结构的选择就是链表，用一个链表来记录信息，释放就删除节点，分配就添加。

1. The program counter contains
   1. the number of times a program has been executed
   2. the address of the CPU instruction that is about to be executed
   3. the amount of memory a program is currently using
   4. the number of CPU instructions a program has executed so far

答案：b，PC（程序计数器）里面放的是下一条要执行的指令。

1. Assume a function foo takes two arguments. When calling foo(arg1, arg2), which is the correct order of operations assuming x86 calling conventions and that foo must allocate stack space(implies that we must save the ebp)?

a. push arg1, push arg2, call foo, push ebp

* 1. push arg1, push arg2, push ebp, call foo
  2. push arg2, push arg1, call foo, push ebp
  3. push arg2, push arg1, push ebp, call foo

答案：c，考点是栈帧，x86，参数从右到左入栈，之后执行call指令，在被调用函数内部的第一条指令是push ebp，接着是将ebp移动到esp，再开辟一定大小的栈空间，故选C。

1. Which of the following statements about alignment within C struct's is true?
   1. Alignment may cause the allocation of unused space.
   2. Alignment is required by all modern processors.
   3. Alignment can help processors access data more efficiently.

a. i and iii only

* 1. i only
  2. ii and iii only
  3. i, ii, and iii

答案：a，参考PPT week04最后一部分。

1. Consider the following code. char a[100];

a[99] = \*((char \*) (((int) &a[0]) + 4))

If integers are 32 bits wide, which of the following values is equal to a[99]?

a. a[0] + 4

* 1. a[3]
  2. a[4]
  3. the integer stored in the bytes a[4], a[5], a[6] and a[7]

答案：c，取出地址后，强制转换为int之后变为加4，便加的是4个单位，一个单位是4个字节，即int的大小，那么这里加4就变为了a[4]。

1. Suppose that, using a tool such as the memory window of Visual C++, we found that a certain set of contiguous memory locations contained the integer

0xC605CD623A8365000000. What could these memory locations hold?

1. the integer 0xC605CD623A8365000000

ii) a string

iii) a CPU instruction

* 1. i only
  2. iii only
  3. i, ii, and iii
  4. i and ii only

答案：c，三种情况都有可能，因为一个位置既可以是整数（32、64位）、也可以是一串字符，更可以是一段指令。

1. Two [code](https://cn.bing.com/dict/search?q=code&FORM=BDVSP6&mkt=zh-cn) fragments are i) and ii) as following:
   1. M= 100000; N=100000; int data[M][N]; for (i = 0; i < M; i++) { for (j = 0 ; j < N; j++) { sum += data[ i ] [ j ];

}

}

* 1. M= 100000; N=100000; int data[M][N]; for (j = 0 ; j < N; j++) { for (i = 0; i < M; i++) { sum += data[ i ] [ j ];

}

}

which one is more efficient (better performance) if you program with Matlab which is column-major order for matrix?

a. Both are very efficient.

* 1. i)
  2. ii)
  3. Both are not efficient.

答案：b，对于多维数组的访问，行访问优于列访问，行访问就是一行一行读取，列访问就是一列一列读取。存储的方式是行存储，行访问的cache命中率高。

1. the advantage of using copying gc including

i) A copying collector is generally more efficient than a non-copying collector

ii) The copying gc can make use of heap memory effectively.

a. i)

* 1. ii)
  2. both of them
  3. none of them

答案：a，复制的办法，需要双倍的空间，具体思路如下：将空间分为等大小的两部分，其中一部分不能使用，另外一部分空间不足时，便将还处于活跃状态的内存块，复制到另外那边，然后整个擦除自己。

1. which of the following are advantages of using statistical sampling to profile programs?
   1. exact run times of all functions can be determined.
   2. code can be instrumented automatically.

iii) the performance impact due to measurement can be minimal

a. i) and iii) only

* 1. ii) and iii) only
  2. i) and ii) only
  3. i), ii), and iii)

答案：不知道，PPT上相关部分并未有此内容。

1. read the following code, and how can we optimize it?

void lower1(char \*s) {

int i; for (i = 0; i < strlen(s); i++) if (s[i] >= 'a' && s[i] <= 'z') s[i] -= ('a' - 'a');

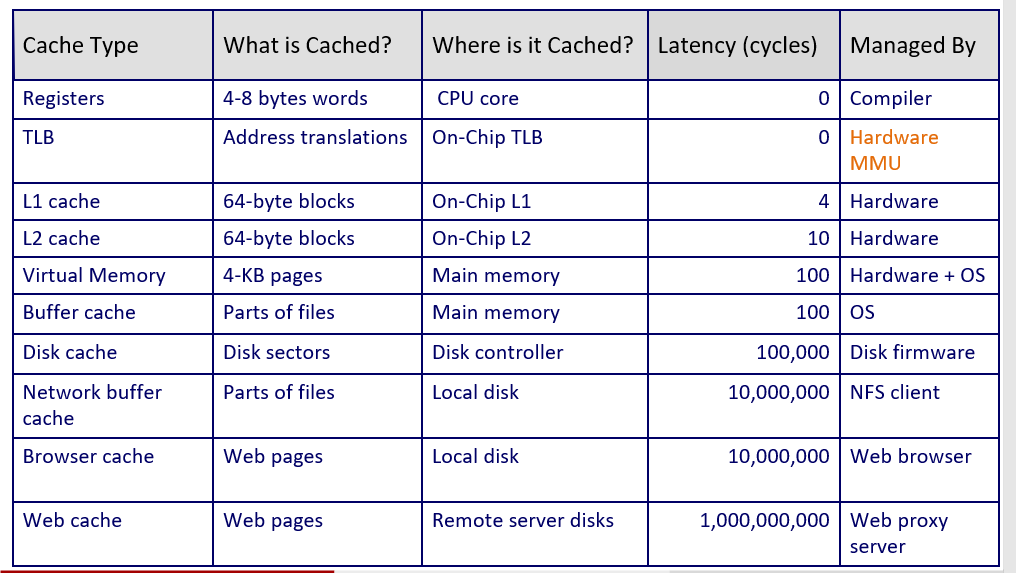
}

* 1. reducing procedure calls
  2. enhancing parallelism
  3. loop splitting
  4. converting to pointer code

答案：a，减少过程调用，因为每次循环都调用了strlen函数，实际上每次调用这个值都是不变的，所以将这个抛离循环就能优化。

1. Which of the following manages the transfer of data between the CPU registers and the cache?
   1. Registry.
   2. Compiler.
   3. Hardware.
   4. Operating System.

答案：b



1. What can Loader do?

i) translate the C code into machine code ii) Resolution

iii) load or map the Executable object file from the disk to memory

a. i) and ii) only.

* 1. i) and iii) only.
  2. i), ii) and iii).
  3. iii) only.

答案：d，i编译器和汇编器做的事，ii是链接器做的事。

1. In IA32 or X86, which exception never returns back to the point where exception happens?
   1. interrupt
   2. trap
   3. fault
   4. Abort

答案：d

解析：

interrupt：总是返回下一条指令

trap：总是返回下一条指令

fault：可能返回当前指令

Abort：不返回

|  |  |
| --- | --- |
| 评阅教师 | 得分 |
|  |  |

## 二、bit operation（本大题共2 小题，各5 分，共10 分）

Now complete the following functions according to the following rules.

Each "Expr" is an expression using ONLY the following:

1. Integer constants 0 through 0xFFFFFFFF inclusive.
2. Function arguments and local variables (no global variables).
3. Some of the problems restrict the set of allowed operators.

You are expressly forbidden to:

1. Use any control constructs such as if, do, while, for, switch, etc.
2. Define or use any macros.
3. Define any additional functions in this file.
4. Call any functions.
5. Use any other operations, such as &&, ||, -, ?, or [] :
6. Use any form of casting.

You may assume that your machine:

1. Uses 2s complement, 32-bit representations of integers.
2. Performs right shifts arithmetically.
3. Has unpredictable behavior when shifting an integer by more than the word size.

/\*

* isNonNegative - return 1 if x >= 0, return 0 otherwise
* Example: isNonNegative(-1) = 0. isNonNegative(0) = 1.
* Legal ops: ! ~ & ^ | + << >>
* Max ops: 6
* Rating: 3

\*/

int isNonNegative(int x) {

return !(x>>31);//整数是32位，右移31只有符号位，若符号位为1就是负数，返回0，其余皆返回1，所以取非。

}

/\*

* isEqual - return 1 if x == y, and 0 otherwise
* Examples: isEqual(5,5) = 1, isEqual(4,5) = 0
* Legal ops: ! ~ & ^ | + << >>
* Max ops: 5
* Rating: 2

\*/ int isEqual(int x, int y) {

return !(x^y);//相等用异或，每位都相同，那就是相同的数，那么异或操作为0，有一位不同，异或都是非0数。

}

|  |  |
| --- | --- |
| 评阅教师 | 得分 |
|  |  |

## 三、memory allocation（本大题共1 小题，共5 分）

Are there any errors in the following program? If there are, please correct them on the right side.

#include <stdio.h>

#include <stdlib.h>

char\* getmemory()

{ char \*p=(char \*) malloc(100); strcpy(p,”hello world”); return &p; //此处返回类型不对，去掉取址符

} int main( )

{

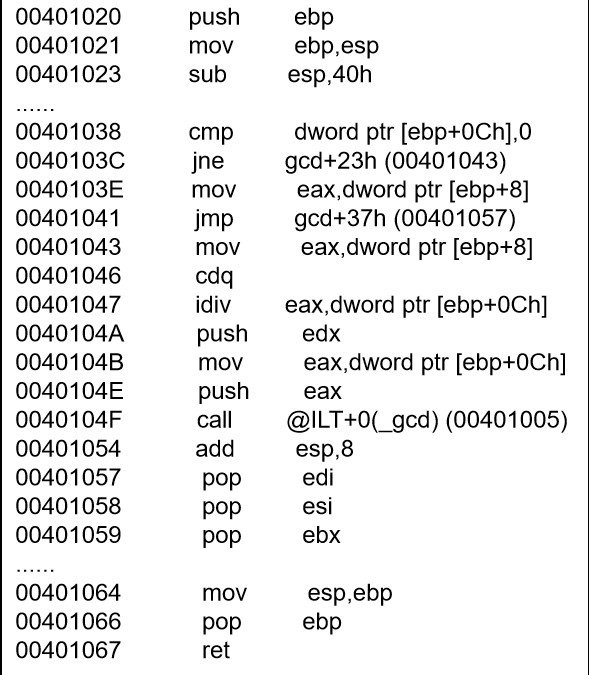
char \*str=NULL; str = getmemory( ); printf(“%s/n”,str); free(str); return 0;

}

|  |  |
| --- | --- |
| 评阅教师 | 得分 |
|  |  |

四、stack discipline（本大题共2 小题，共15 分）

Consider the following C code and assembly code for a recursive function:

 int gcd(int a, int b)

{

if(!b) { return a;

}

return gcd(b, a % b);

}

Imagine that a program makes the procedure call gcd(213, 18). Also imagine that prior to the invocation, the value of esp is 0xffff1000—that is, 0xffff1000 is the value of esp immediately before the execution of the call instruction.

1. (12 points) Note that the call gcd(213, 18) will result in the following function invocations: gcd(213,18), gcd(18, 15), gcd(15, 3), and gcd(3, 0). Using the provided code and your knowledge of IA32 stack discipline, fill in the stack diagram with the values that would be present immediately before the execution of the leave instruction for gcd(15, 3). Supply numerical values wherever possible, and cross out each blank for which there is insufficient information to complete with a numerical value.

|  |  |  |
| --- | --- | --- |
| **Address no.** | **Value** | **Description** |
| 0xffff1008 | unknown | unknown |
| 0xffff1004 | 18 | 压入参数b |
| 0xffff1000 | 213 | 压入参数a |
| 0xffff0ffc | xxx | 返回地址 |
| 0xffff0ff8 | old ebp | 调用之前的ebp |
| 0xffff0ff4 |  |  |
| 0xffff0ff0 |  |  |
| **Address no.** | **Value** | **Description** |
| 0xffff0fec |  |  |
| 0xffff0fe8 |  |  |
| 0xffff0fe4 |  |  |
| 0xffff0fe0 |  |  |
| 0xffff0fdc |  |  |
| 0xffff0fd8 |  |  |
| 0xffff0fd4 |  |  |
| 0xffff0fd0 |  |  |
| 0xffff0fcc |  |  |
| 0xffff0fc8 |  |  |
| 0xffff0fc4 |  |  |
| 0xffff0fc0 |  |  |
| 0xffff0fbc |  |  |
| 0xffff0fb8 |  | 40h cc |
| 0xffff0fb4 | 15 | push edx |
| 0xffff0fb0 | 18 | push eax |

解析（只有最后两位不同）：

00处是调用call之前的esp指向的位置，而call之前便是压入参数，所以那两格是放参数，参数之前的栈帧是不能够确定的，所以是unknown。

08：unknown，理由是之前的栈情况不清楚

04：18，参数从右往左入栈，先压入18

00：213，参数从右往左入栈，后压入213

fc：返回地址（call之后入栈），call语句的下一行，此刻不清楚

f8：main函数栈帧的ebp值，由汇编代码中的push ebp压入的

b8：40h cc sub esp 40h开辟空间，使得esp向下移动了40h（16进制），f8-40=b8

接着递归函数，又开始调用该函数重复参数、返回值入栈的过程

b4：15 push edx右边第一个参数入栈

b0：18 push eax右边得个参数入栈

1. (3 points) What are the values of esp and ebp immediately before the execution of the ret instruction for gcd(15, 3)?

ebp = 0xffff0fa8

esp = 0xffff0f5c

解析：

gcd（15,3）ret指令之前，做了两件事，mov esp,ebp;pop ebp;

mov esp,ebp 使得栈顶指针回到了，栈帧的栈底处，也就是栈顶指向了old ebp

pop ebp pop之前esp指向了old ebp，pop就将ebp指向了old ebp，也使得esp指向了前一个位置。

所以在ret之前，ebp指向了gcd(18,15)的栈帧的底部，而esp指向了gcd(15,3)的返回地址处。

接着上面画栈帧：

ac xxx 返回地址,call的下一行，举个例子上述片段call下一条00401054，只不过不知道这个片段是哪个函数的栈帧

a8 f8 old ebp

68 cc 40h cc

64 3 参数

60 15 参数

5c xxx 返回地址

58 a8 old ebp

mov使得esp先指向了58，pop使得esp上移，指向了5c，并使得ebp指向了a8

故最后结果如上。

## 五、performance optimization（本大题共2 小题，共12 分）

1. (6 points) We are considering an enhancement to the processor of a web server. Then new CPU is 10 times faster on search queries than the old processor. The old processor is busy with search queries 60% of the time, what is the speedup gained by integrating the enhanced CPU?

加速比：1/(1-P+P/S)=1/(1-0.6+0.6/10)=2.17

1. (6 points) Read the following two code fragments, func1(x) and func2(x) . Do func1(x) and func2(x) have same behavior in any circumstances? Give an example of f(int) to show that func1(x) and func2(x) have the same behavior. If their behaviors are not same in some cases, give another example of f(int).

**int f(int);** **int func1(x) {** **return f(x) + f(x) + f(x) + f(x);**

**}**

**int func2(x) {** **return 4\*f(x);**

**}**

两个函数虽然都返回了相同的结果，但是，行为上却不是完全相同。例子如下：

当传入的参数是一个全局变量时，第一个函数会修改4次全局变量的值，而第二个函数只会做一次修改，即两种函数造成的全局变量的值不一样。

## 六、cache（本题共8 分）

Read the code as following:

int i

int sum=0; int data[100000]; for (i = 0; i < 100000; i += stride) sum += data[i];

Assume that size of int is 4-byte and the cache line (block) length is 32-byte.

If the stride is 1,2,4,8 , what is their hit ratio and miss ratio? Fill the blanks in the table and give your explain.

|  |  |  |
| --- | --- | --- |
| **Stride** | **Hit ratio** | **Miss ratio** |
| 1 | 0.875 | 0.125 |
| 2 | 0.75 | 0.25 |
| 4 | 0.5 | 9.5 |
| 8 | 0 | 1 |

cache里能放8个整数，故当i=1时，每8次便会产生一次cache未命中，总共访问了1000次，1000/8=125，故产生了125次未命中，所以未命中率为125/1000=0.128，命中率为875/1000=0.872。

当i=2时，每4次便产生一次cache未命中，总共访问500次，500/4=125，故产生了125次未命中，所以未命中率为125/500=0.25，故命中率为0.75.

当i=4时，每2次便产生一次cache未命中，总共访问250次，250/2=125，故产生了125次未命中，所以未命中率为125/250=0.5，故命中率为0.5.

挡i=8时，每次都不会命中，故未命中率为1，命中率为0.

## 七、linker（本题共10 分)

There is a program built by the following two c files: main.c and swap.c. Reading the source code, and please give a description on when linking, what should linker do in order to merge them. And drawing a figure is suggested.

/\*main.c\*/ extern int shared; int main()

{

int a=100; swap(&a,&shared);

}

/\*swap.c\*/ int shared=1; void swap(int\* a, int\* b)

{

\*a^=\*b^=\*a^=\*b;

}

参考week10 ppt，自行总结。