

Andrew login ID: _____

Full Name: _____

Recitation Section: _____

CS 15-213, Spring 2009

Exam 1

Tuesday, February 24, 2009

Instructions:

- Make sure that your exam is not missing any sheets, then write your full name, Andrew login ID, and recitation section (A–J) on the front.
- Write your answers in the space provided for the problem. If you make a mess, clearly indicate your final answer.
- The exam has a maximum score of 100 points.
- The problems are of varying difficulty. Try to solve the problems as quickly and thoroughly as possible. The easier problems are worth fewer points than the harder problems.
- This exam is OPEN BOOK. You may use any books or notes you like. No calculator or other electronic devices are allowed.
- Good luck!

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

1 (16):
2 (22):
3 (13):
4 (13):
5 (22):
6 (14):
TOTAL (100):

Problem 1. (16 points):

Consider a new floating point format that follows the IEEE spec you should be familiar, except with 3 exponent bits and 2 fraction bits (and 1 sign bit). Fill in all blank cells in the table below. If, in the process of converting a decimal number to a float, you have to round, write the rounded value next to the original decimal as well.

Description	Decimal	Binary Representation
Bias		-----
Smallest positive number		
Lowest finite		
Smallest positive normalized		
-----	$-\frac{7}{16}$	
	$\frac{5}{4}$	
		1

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Problem 2. (22 points):

Consider the C code written below and compiled on a 32-bit Linux system using GCC.

```
struct s1
{
    short x;
    int y;
};
```

```
struct s2
{
    struct s1 a;
    struct s1 *b;
    int x;
    char c;
    int y;
    char e[3];
    int z;
};
```

```
short fun1(struct s2 *s)
{
    return s->a.x;
}
```

```
void *fun2(struct s2 *s)
{
    return &s->z;
}
```

```
int fun3(struct s2 *s)
{
    return s->z;
}
```

```
short fun4(struct s2 *s)
{
    return s->b->x;
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

a) What is the size of `struct s2`?

b) How many bytes are wasted for padding?

Assignment Project Exam Help

<https://eduassistpro.github.io/>

You may use the rest of the s

Nothing written below this line will be graded.

Add WeChat `edu_assist_pro`

c) Which of the following correspond to functions fun1, fun2, fun3, and fun4?

```
push    %ebp
mov     %esp,%ebp
mov     0x8(%ebp),%eax
add     $0x1c,%eax
pop     %ebp
ret
```

ANSWER: _____

```
push    %ebp
mov     %esp,%ebp
mov     0x8(%ebp),%eax
mov     0x8(%eax),%eax
movswl  (%eax),%eax
pop     %ebp
ret
```

ANSWER: _____

```
push    %ebp
mov     %esp,%ebp
mov     0x8(%ebp),%eax
mov     0x1c(%eax),%eax
pop     %ebp
ret
```

ANSWER: _____

```
push    %ebp
mov     %esp,%ebp
mov     0x8(%ebp),%eax
movswl  (%eax),%eax
pop     %ebp
ret
```

ANSWER: _____

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

- d) Assume a variable is declared as `struct s2 myS2;` and the storage for this variable begins at address `0xbfb2ffc0`.

```
(gdb) x/20w &myS2
0xbfb2ffc0: 0x0000000f  0x000000d5  0xbfb2ffe8  0x00000000
0xbfb2ffd0: 0xb7f173ff  0x0000012c  0xbf030102  0x0000000c
0xbfb2ffe0: 0xb7e2dfd5  0xb7f15ff4  0xbfb30012  0x000000f3
0xbfb2fff0: 0xb7e2e0b9  0xb7f15ff4  0xbfb30058  0xb7e1adce
0xbfb30000: 0x00000001  0xbfb30084  0xbfb3008c  0xbfb30010
```

Fill in all the blanks below.

HINTS: Label the fields. Not all 20 words are used. Remember endianness!

What would be returned by:

`fun1(&myS2) = 0x_____`

`fun2(&myS2) = 0x_____`

`fun3(&myS2) = 0x_____`

`fun4(&myS`

What is the value of:

`myS2.b->y = 0x_____`

`myS2.a.y = 0x_____`

`myS2.z = 0x_____`

`myS2.e[1] = 0x_____`

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Problem 3. (13 points):

Given the memory dump and disassembly from GDB on the next page, fill in the C skeleton of the function switchfn:

```
int switchfn(int a, long b) {  
  
    int y = 0, x = _____;  
    switch (a * b) {  
  
        case 1:  
            return 24;  
  
        case 6:  
  
            a = _____;  
            return a;  
  
        case 0:  
            return a + b;  
  
        case 4:  
            x = a;  
            y *= b;  
            break;  
  
        case ____:  
            a = y == x;  
  
        case 3:  
  
            b = y ____ x;  
  
        case 5:  
  
            return a ____ b;  
    }  
  
    return x == y;  
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

There may be a few instructions you haven't seen before in this assembly dump. `data16` is functionally equivalent to `nop`. `setcc` functions similarly to `jcc` except it will set its operand to 1 or 0 instead of jumping or not jumping, respectively. `cqto` is the 64-bit equivalent of `cld`.

```
(gdb) x/7xg 0x4005c0
```

```
0x4005c0 <_IO_stdin_used+8>:    0x00000000004004a1    0x0000000000400494
0x4005d0 <_IO_stdin_used+24>:   0x00000000004004ac    0x00000000004004b4
0x4005e0 <_IO_stdin_used+40>:   0x00000000004004a5    0x00000000004004bc
0x4005f0 <_IO_stdin_used+56>:   0x000000000040049a
```

```
0x0000000000400476 <switchfn+0>:    mov     $0x0,%ecx
0x000000000040047b <switchfn+5>:    mov     $0xdeadbeef,%edx
0x0000000000400480 <switchfn+10>:   movslq  %edi,%rax
0x0000000000400483 <switchfn+13>:   imul    %rsi,%rax
0x0000000000400487 <switchfn+17>:   cmp     $0x6,%rax
0x000000000040048b <switchfn+21>:   ja      0x4004c5 <switchfn+79>
0x000000000040048d <switchfn+23>:   jmpq    *0x4005c0(,%rax,8)
0x0000000000400494 <switchfn+30>:   mov     $0x18,%eax
0x0000000000400499 <switchfn+35>:   retq
0x000000000040049a <switchfn+38>:   lea     (%rdi,%rsi,4),%eax
0x000000000040049d <switchfn+39>:   data16
0x0000000000400400
0x0000000000400400
0x0000000000400400
0x00000000004004a1 <switchfn+43>:   lea     (%rdi,%rsi,1),%eax
0x00000000004004a4 <switchfn+46>:   retq
0x00000000004004a7 <switchfn+49>:   mov     %eax,%edx
0x00000000004004a7 <switchfn+49>:   imul    %esi,%ecx
0x00000000004004aa <switchfn+52>:   jmp     0x4004c5 <switchfn+79>
0x00000000004004ac <switchfn+54>:   cmp     %edx,%ecx
0x00000000004004ae <switchfn+56>:   sete    %al
0x00000000004004b1 <switchfn+59>:   movzbl  %al,%edi
0x00000000004004b4 <switchfn+62>:   cmp     %edx,%ecx
0x00000000004004b6 <switchfn+64>:   setl    %al
0x00000000004004b9 <switchfn+67>:   movzbl  %al,%esi
0x00000000004004bc <switchfn+70>:   movslq  %edi,%rax
0x00000000004004bf <switchfn+73>:   cqto
0x00000000004004c1 <switchfn+75>:   idiv    %rsi
0x00000000004004c4 <switchfn+78>:   retq
0x00000000004004c5 <switchfn+79>:   cmp     %ecx,%edx
0x00000000004004c7 <switchfn+81>:   sete    %al
0x00000000004004ca <switchfn+84>:   movzbl  %al,%eax
0x00000000004004cd <switchfn+87>:   retq
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat [edu_assist_pro](#)

Problem 4. (13 points):

The function below is hand-written assembly code for a sorting algorithm. Fill in the blanks on the next page by converting this assembly to C code.

```
.globl mystery_sort      # exports the symbol so other .c files
                          # can call the function

mystery_sort:
    jmp     loop1_check

loop1:
    xor     %rdx, %rdx
    mov     %rsi, %rcx
    jmp     loop2_check

loop2:
    mov     (%rdi, %rcx, 8), %rax
    cmp     %rax, (%rdi, %rdx, 8)
    jg      loop2_check
    mov     %rcx, %rdx

loop2_check:
    dec     %rcx
    test    %rcx, %rcx
    jnz     loop2

    dec     %rsi
    mov     (%rdi, %rsi, 8), %rax
    mov     (%rdi, %rdx, 8), %rcx
    mov     %rcx, (%rdi, %rsi, 8)
    mov     %rax, (%rdi, %rdx, 8)

loop1_check:
    test    %rsi, %rsi
    jnz     loop1

    ret
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

```

void mystery_sort (long* array, long len)
{
    long a, b, tmp;

    while (_____ > _____)
    {

        a = _____;

        for (b = _____; b > _____; b--)
        {

            if (array[_____] > array{_____})
            {

                _____ = _____;

            }

        }

        len--;

        tmp = array[_____]

        array[_____]

        array[_____] = tmp;

    }
}

```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

Problem 5. (22 points):

Circle the correct answer.

1. What sequence of operations does the leave instruction execute?

- (a) `mov %ebp, %esp`
`pop %ebp`
- (b) `pop %ebp`
`mov %ebp, %esp`
- (c) `pop %esp`
`mov %ebp, %esp`
- (d) `push %ebp`
`mov %esp, %ebp`

2. Who is responsible for storing the return address of a function call?

- (a) the caller
- (b) the callee
- (c) the kernel
- (d) the CPU

3. On what variable types does

- (a) signed types
- (b) unsigned types
- (c) signed and unsigned types
- (d) C does not perform logical right shifts

4. What is the difference between the rbx and the ebx register on an x86_64 machine?

- (a) nothing, they are the same register
- (b) ebx refers to only the low order 32 bits of the rbx register
- (c) they are totally different registers
- (d) ebx refers to only the high order 32 bits of the rbx register

5. Which of the following is the name for the optimization performed when you pull code outside of a loop?

- (a) code motion
- (b) loop expansion
- (c) dynamic programming
- (d) loop unrolling

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

6. On 32-bit x86 systems, where is the value of `%ebp` saved in relation to the current value of `%ebp`? (Assume a pointer size of 32 bits.)
- (a) there is no relation between where the current base pointer and old base pointer are saved.
 - (b) `old ebp = (ebp - 4)`
 - (c) `old ebp = (ebp + 4)`
 - (d) `old ebp = (ebp)`
7. Which of the following `mov` instructions is invalid?
- (a) `mov %esp, %ebp`
 - (b) `mov $0xdeadbeef, %eax`
 - (c) `mov (0xdeadbeef), %esp`
 - (d) `mov $0xdeadbeef, 0x08048c5f`
 - (e) `mov %ebx, 0x08048c5f`
8. In C, the result of shifting a value by greater than its type's width is:
- (a) illegal
 - (b) undefined
 - (c) 0
 - (d) Encouraged by the standard
9. Extending the stack can
- (a) swapping the base pointer and the stack pointer
 - (b) subtracting a value from your stack pointer
 - (c) adding a value to your stack pointer
 - (d) executing the `ret` instruction
10. 64-bit systems can support 32-bit assembly code
- (a) TRUE
 - (b) FALSE
11. Assuming the register `%rbx` contains the value `0xfaaafbbbfcccfddd`, which instruction would cause the register `%rdi` to contain the value `0x00000000fcccfddd`?
- (a) `movl %ebx, %rdi`
 - (b) `movslq %ebx, %rdi`
 - (c) `movzbl %ebx, %rdi`
 - (d) `leal %ebx, %rdi`

Problem 6. (14 points):

Throughout this question, remember that it might help you to draw a picture. It helps us see what you're thinking when we grade you, and you'll be more likely to get partial credit if your answers are wrong.

Consider the following C code:

```
void foo(int a, int b, int c, int d) {
    int buf[16];
    buf[0] = a;
    buf[1] = b;
    buf[2] = c;
    buf[3] = d;
    return;
}

void bar() {
    foo(0x15213, 0x18243, 0xdeadbeef, 0xcafebabe)
}
```

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

When compiled with default options (32-bit), it gives the following assembly:

```
00000000 <foo>:
  0:  55                push    %ebp
  1:  89 e5             mov     %esp,%ebp
  3:  83 ec 40          sub     $0x40,%esp

  6:  8b 45 08          mov     _____(%ebp),%eax //temp = a;
  9:  89 45 c0          mov     %eax,-0x40(%ebp) //buf[0] = temp;

  c:  8b 45 0c          mov     _____(%ebp),%eax //temp = b;
  f:  89 45 c4          mov     %eax,-0x3c(%ebp) //buf[1] = temp;

 12:  8b 45 10          mov     _____(%ebp),%eax //temp = c;
 15:  89 45 c8          mov     %eax,-0x38(%ebp) //buf[2] = temp;

 18:  8b 45 14          mov     _____(%ebp),%eax //temp = d;
 1b:  89 45 cc          mov     %eax,-0x34(%ebp) //buf[3] = temp;
 1e:  c9                leave   %ebp
 1f:  c3                ret
```

```
00000020 <bar>:
 20:  55                push    %ebp
 21:  89 e5             mov     %esp,%ebp
 23:  83 ec 10          sub     $0x10,%esp
 26:  c7 44 24 0c be ba fe ca movl     $0xcafebabe,0xc(%esp)
 2e:  c7 44 24 08 ef be ad de movl     $0xdeadbeef,0x8(%esp)
 36:  c7 44 24 04 43 82 01 00 movl     $0x18243,0x4(%esp)
 3e:  c7 04 24 13 52 01 00    movl     $0x15213, (%esp)
 45:  e8 fc ff ff ff    call    foo
 4a:  c9                leave   %ebp
 4b:  c3                ret
```

- a) Very briefly explain what purpose is served by the first three lines of the disassembly of foo (just repeating the code in words is not sufficient). No more than one sentence should be necessary here.
- b) Note that in foo (C version), each of the four arguments are accessed in turn. The assembly dump of foo is commented to show where this is done. Recall that the current `%ebp` value points to where the pushed old base pointer resides, and immediately above that is the return address from the function call. Write into the gaps in the disassembly of foo the offsets from `%ebp` needed to access each of the four arguments a, b, c, and d. (Hint: Look at how they are arranged in bar before the call.)

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro

GCC has a compile option called `-fomit-frame-pointer`. When given this flag in addition to the previous flags, the function `foo` is compiled like this:

```

00000000 <foo>
83 ec 40      sub     $0x40,%esp

8b 44 24 44    mov     _____(%esp),%eax //temp = a;
89 04 24      mov     %eax,(%esp)          //buf[0] = temp;

8b 44 24 48    mov     _____(%esp),%eax //temp = b;
89 44 24 04    mov     %eax,0x4(%esp)       //buf[1] = temp;

8b 44 24 4c    mov     _____(%esp),%eax //temp = c;
89 44 24 08    mov     %eax,0x8(%esp)       //buf[2] = temp;

8b 44 24 50    mov     _____(%esp),%eax //temp = d;
89 44 24 0c    mov     %eax,0xc(%esp)       //buf[3] = temp;
83 c4 40      add     $0x40,%esp
c3           ret

```

Assignment Project Exam Help

- c) What is the difference between the first few lines of `foo` in the first compilation and in this compilation? What does this mean about the picture.)

<https://eduassistpro.github.io/>

Add WeChat `edu_assist_pro`

- d) Note what has changed in how the arguments `a`, `b`, `c`, `d` and the stack-allocated buffer are accessed: they are now accessed relative to `%esp` instead of `%ebp`. Considering that the arguments are in the same place when `foo` starts as last time, and recalling what has changed about the stack this time around (note: the pushed return address is still there!), fill in the blanks on the previous page to correctly access the function's arguments.
- e) Consider what the compiler has done: `foo` is now using its stack frame without dealing with the base pointer at all... and, in fact, all functions in the program compiled with `-fomit-frame-pointer` also do this. What is a benefit of doing this? (0-point bonus question: What is a drawback?)

Assignment Project Exam Help

<https://eduassistpro.github.io/>

Add WeChat edu_assist_pro