UCLA Computer Science 33 (Fall 2016)
Midterm 1
100 minutes, 100 points, open book, open notes.
Use a separate sheet of paper for each answer, except for problem 1 where you should simply write the answer on your copy of the exam.
Put a big problem number at each sheet's top.
Turn in your sheets in increasing numeric order.

Name: [redacted] Student ID: [redacted]

1	2	3	4	5	6	7	8	9	10
14	4	2	2	6	0/4	7	0	0	0

39

Consider the following C program and x86-64 assembly-language code in gcc -S format:

```
int a (int m) { return m << 31 << 2; }
int b (int m) { return m >> 31 >> 2; }
int c (int m, int n) { return m >> n >> 2; }
int d (int m, int n) { return m << n >> 2; }
int e (int m, int n) { return n >> m >> 2; }
int f (int m, int n) { return n << m >> 2; }
int g (int m, int n) { return m >> 2 >> n; }
int h (int m, int n) { return m << 2 >> n; }
int i (int m, int n) { return n >> 2 >> m; }
int j (int m, int n) { return n << 2 >> m; }
```

```
L1:
        movl    %edi, %eax
        movl    %esi, %ecx
        sarl    $2, %eax
        sarl    %cl, %eax
        ret
L2:
        movl    %edi, %ecx
        sarl    %cl, %esi
        movl    %esi, %eax
        sarl    $2, %eax
        ret
L3:
        movl    %esi, %ecx
        sarl    %cl, %edi
        movl    %edi, %eax
        sarl    $2, %eax
        ret
L4:
        movl    %edi, %eax
        sarl    $31, %eax
        ret
L5:
        xorl    %eax, %eax
        ret
L6:
        leal    0(,%rdi,4), %eax
        movl    %esi, %ecx
        sarl    %cl, %eax
        ret
L7:
        leal    0(,%rsi,4), %eax
        movl    %edi, %ecx
        sarl    %cl, %eax
        ret
L8:
        movl    %esi, %eax
        movl    %edi, %ecx
        sarl    $2, %eax
        sarl    %cl, %eax
        ret
L9:
        movl    %edi, %ecx
        sall    %cl, %esi
        movl    %esi, %eax
        sarl    $2, %eax
        ret
```

This code is covered in questions 1 through 5 on the last page.

a b c d e f g h i j

1 2 3 4 5 6 7 8 9

Consider also the following x86-64 assembly-language code, in gcc -S format.

```
F:
        movl    %edi, %eax
        andl    $1, %eax
        ret

G:
        pushq   %rbp
        leaq    1(%rdi), %rax
        pushq   %rbx
        subq    $8, %rsp
        cmpq    $1, %rax
        jbe     .L4
        movq    %rdi, %rbx
        xorl    %ebp, %ebp
.L3:
        movq    %rbx, %rdi
        sarq    %rdi
        xorq    %rbx, %rdi
        sarq    $2, %rbx
        call    f
        movzbl  %al, %eax
        addl    %eax, %ebp
        leaq    1(%rbx), %rax
        cmpq    $1, %rax
        ja      .L3
.L6:
        addq    $8, %rsp
        movl    %ebp, %eax
        popq    %rbx
        popq    %rbp
        ret
.L4:
        xorl    %ebp, %ebp
        jmp     .L6
```

Hint: 'sarq %rdi' is equivalent to 'sarq $1, %rdi'.

This code is covered in questions 6 through 10 on the last page.

[this column is intentionally blank]

1 (18 minutes). Each assembly-language function (L1 through L9) corresponds to a C function (a through j). Write the letter of the C function next to the corresponding assembly-language function. Since there are ten C functions and nine assembly-language functions, one C function should be unused.

2 (5 minutes). Give assembly-language code for the C function that was unused in the previous question.

3 (10 minutes). The source code contains 3 instances of '<< 2' and 7 instances of '>> 2'. However, the assembly-language code contains no instances of 'sall $2,...' and only 5 instances of 'sarl $2,...'. Explain why the compiler can omit each "missing" shift instruction.

4 (10 minutes). Suppose we replace each 32-bit instruction in the above assembly-language code by the corresponding 64-bit instruction. For example, we replace "movl %esi, %eax" by "movq %rsi, %rax" and we replace "sarl %cl, %eax" by "sarq %cl, %rax". What correctness bugs, if any, would this introduce in L1 through L9 as implementations of their corresponding C functions? Briefly explain. Do not worry about performance.

5 (6 minutes). Explain why the following code is not a valid alternate implementation for the C function that matches L8:

```
L8-wrong:
        leal    2(%rdi), %ecx
        movl    %esi, %eax
        sarl    %cl, %eax
        ret
```

6a (2 minutes): Explain from the caller's point of view what F does.

6b (4 minutes): Give C source code that corresponds to F. Briefly justify the types that you use.

7a (5 minutes): Suppose we remove all the pushq and popq instructions from G. Explain what (if anything) could go wrong, from the point of view of G's caller.

7b (10 minutes): Suppose we remove the addq and subq instructions from G. Explain what (if anything) could go wrong, from the point of view of G's caller, and explain whether your answer depends on the internal behavior of F.

8 (12 minutes): Speed up G by inlining the body of F; that is, assume the internal behavior of F and use this assumption to minimize the number of instructions executed by G. Your modified version of G should not call F.

Suppose you are working for a startup called Reduced Intel, which has licensed the x86-64 architecture (and the name "Intel"!) from Intel, and which is building processors that are simpler and faster than x86-64 processors.

9 (6 minutes): You are considering removing the 'call' instruction. Rewrite G so that it calls F according to the usual x86-64 calling conventions, but does not use the 'call' instruction. Do not assume anything about F's internal behavior.

10 (12 minutes): You are also considering removing all jump instructions; that is, you will remove unconditional jumps 'jmp' and all conditional jumps like 'jbe'. Rewrite G so that it does not use any jump or call instructions. As before, your rewrite should not assume anything about F's internal behavior.