UCLA Computer Science 33 (Fall 2018)
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:____________________________________
----+----+----+----+----+----+----+----+
1   |2   |3   |4   |5   |6   |7   |8   |
    |    |    |    |    |    |    |    |
    |    |    |    |    |    |    |    |
----+----+----+----+----+----+----+----+
```

Consider the x86-64 assembly-language code (in gcc -S format) and the C functions on the back side of this exam.

1 (20 minutes). For each assembly-language function (L1 through L14) that corresponds to a C function (a through t), write the letter of the C function next to the corresponding assembly-language function. Note that some assembly functions correspond to more than one C-language function, and some correspond to zero C-language functions.

2 (8 minutes). For each assembly-language function (L1 through L14) that does not correspond to any C-language function, write the source code of a C-language function that it *could* correspond to.

3a (10 minutes). For some arguments, L7 and L10 are equivalent (i.e., they return the same results), and for other arguments they are not equivalent. Give an example argument where they agree, and an example argument where they disagree. Give the exact set of arguments where L7 and L10 agree and why.

3b (12 minutes). Trace through assembly language function L5 with the integer argument -300, and state briefly what each instruction does.

4 (12 minutes). Briefly explain L5. Why is it a correct translation of the corresponding C source code? What does this function do, at a high level?

Consider the following objdump -d output:

```
0000000000000000 <f>:
   0:   48 39 d7                cmp    %rdx,%rdi
   3:   7d 2b                   jge    30 <f+0x30>
   5:   48 89 f9                mov    %rdi,%rcx
   8:   31 c0                   xor    %eax,%eax
   a:   48 f7 d1                not    %rcx
   d:   0f 1f 00                nopl   (%rax)
  10:   40 f6 c7 01             test   $0x1,%dil
  14:   49 89 c8                mov    %rcx,%r8
  17:   4c 0f 44 c7             cmove  %rdi,%r8
  1b:   48 01 f7                add    %rsi,%rdi
  1e:   48 29 f1                sub    %rsi,%rcx
  21:   4c 31 c0                xor    %r8,%rax
  24:   48 39 fa                cmp    %rdi,%rdx
  27:   7f e7                   jg     10 <f+0x10>
  29:   c3                      retq
  2a:   66 0f 1f 44 00 00       nopw   0x0(%rax,%rax,1)
  30:   31 c0                   xor    %eax,%eax
  32:   c3                      retq
```

5 (15 minutes). Reverse-engineer this code, and write a C function that has the same effect.

6 (3 minutes). How often is the instruction at offset 2a executed and why is it there?

7 (10 minutes). This implementation has duplicate cmp instructions at offsets 0 and 24, each followed by a conditional branch. Change the assembly code so that it has just one cmp instruction and one conditional branch, thus making the program a bit shorter.

8 (10 minutes). This implementation has a conditional-move instruction. Rewrite that part so that it doesn't use a conditional of any kind. If you combine (7) and (8), your implementation should have just one conditional jump instruction and no other conditional instructions.

```
L1:
        xorl    %eax, %eax
        ret
L2:
        movq    %rdi, %rax
        notq    %rax
        ret
L3:
        movl    %edi, %ecx
        sarq    %cl, %rdi
        movzbl  %dil, %eax
        ret
L4:
        movq    %rdi, %rax
        sarq    %rax
        xorq    %rdi, %rax
        ret
L5:
        testq   %rdi, %rdi
        leaq    255(%rdi), %rax
        movq    %rdi, %rdx
        cmovns  %rdi, %rax
        sarq    $63, %rdx
        shrq    $56, %rdx
        addq    %rdx, %rdi
        sarq    $8, %rax
        movzbl  %dil, %edi
        subq    %rdx, %rdi
        subq    %rdi, %rax
        ret
L6:
        movq    %rdi, %rax
        ret
L7:
        testq   %rdi, %rdi
        leaq    255(%rdi), %rax
        cmovns  %rdi, %rax
        sarq    $8, %rax
        ret
L8:
        movq    %rdi, %rax
        movl    %edi, %ecx
        sarq    %cl, %rax
        ret
L9:
        movq    %rdi, %rax
        sarq    $63, %rax
        ret
```

```
L10:
        movq    %rdi, %rax
        sarq    $8, %rax
        ret
L11:
        movzbl  %dil, %eax
        movl    %edi, %ecx
        sarq    %cl, %rax
        ret
L12:
        leaq    (%rdi,%rdi), %rax
        ret
L13:
        movl    $1, %eax
        ret
L14:
        imulq   %rdi, %rdi
        movq    %rdi, %rax
        ret
```

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

```
long a (long m) { return 0; }
long b (long m) { return -1 - m; }
long c (long m) { return m < 0 ? -1 : 0; }
long d (long m) { return (m & 255) >> m; }
long e (long m) { return m / 256; }
long f (long m) { return m / 256 - m % 256; }
long g (long m) { return m << 63 << 1; }
long h (long m) { return m >> 63 >> 1; }
long i (long m) { return m >> 8; }
long j (long m) { return m & m; }
long k (long m) { return m * m; }
long l (long m) { return m + m; }
long m (long m) { return m - m; }
long n (long m) { return m / m; }
long o (long m) { return m >> m; }
long p (long m) { return m ^ m; }
long q (long m) { return m | m; }
long r (long m) { return (m >> m) & 255; }
long s (long m) { return m >> (m & 255); }
long t (long m) { return ~m; }
```

CS33 Fall 2018 Midterm 1 Answers

1) (on test sheet)

2) L6:

```
long L6(long m){
    return m;
}
```

L4:

```
long L4(long m){
    return (m^(m>>1));
}
```

L9:

```
long L9(long m){
    return m>>63;
}
```

L13

```
long L13(long m){
    return 1;
}
```

3)(a) Agree Ex:

If m=3, both L7 & L10 return $3 \cdot 2^8$

Disagree Ex:

If m=-3, L7 returns 2^8 but L10 returns $3 \cdot 2^8$

Agreement Set:

L7 agrees with L10 when $m \geq 0$ because the condition in L7 will only return m>>8 when it is unsigned, which is determined by testq whereas L10 returns m>>8 always.

3(b) L5:

//tests -300, sets SF b/c rdi is signed 111...1011010100 (55)

// stores address of least sig bit of -300 into rax

// sets rdx = -300

// skips cmovns line b/c it is signed

// sets rdx = 1111...1 which is the (due to the) sign bit

// sets rdx = 00...011111111 (56, 8)

// sets rdi = -300 + 00...011111111

// shifts rax to the right, making it equal to 0

// set rdi = least sig bit = 1

// rdx = rdi - rdx = ...

// rax = rax - rdi

// returns rax = 0 - 1 = -1

4) L5 matches with C because the assembly code's output matches with the C-level code's outputs depending on the value of the arguments. This function just tells us that if m<0, return the all 1 masking bit, otherwise it returns the all 0 masking bit which can be used for condition-checking w/out using if statements.

5) foo(a,b,c){
    if (a≥c)
        return 0;
    while(c≥a+b)