UCLA Computer Science 33 (Spring 2015)
Midterm
108 minutes total, open book, open notes
Questions are equally weighted (12 minutes each)

Name: [redacted] Student ID: [redacted]

1	2	3	4	5	6	7	8	9	sum
12	0	2	0	0	6	12	10	9	51

1 (12 minutes). Which integer constants can a single x86 leal instruction can multiply an arbitrary integer N by? The idea is that one puts N into a register, executes the leal instruction, and the bottom 32 bits of N*K will be put into some other register, where K is a constant. For which values of K can this be done? For each such value, show an leal instruction that implement that value.

2 (12 minutes). On the x86-64, what's the fastest way to reverse each 8-bit byte in a 64-bit unsigned integer? For example, given the input integer 0x0123456789abcdef, we want to compute 0x80c4a2e691d5b3f7; this is because 0x01 is binary 00000001 and reversing it yields binary 10000000 which is 0x80, and similarly the bit-reverse of 0x23 is 0xc4, and so forth until the bit-reverse of 0xef is 0xf7. Write the code in C, and estimate how many machine instructions will be generated (justify your estimate).

3 (12 minutes). On the x86, there is no 'pushl %eip' instruction. Suppose you want to push the instruction pointer onto the stack anyway. What's the best way to do it? If your method takes three instructions A, B, C, the value pushed onto the stack should be the address of D, the next instruction after C.

4 (12 minutes). Explain two different methods that GDB can implement its 'fin' command (which finishes execution of the current function), one method with hardware breakpoints and one without. For each method, say what happens if the current function calls another function via tail recursion.

5 (12 minutes). Consider the following C functions and their translations to x86 code.

```
int f (int *p, long *q) {
  ++*p;
  ++*q;
  return *p;
}

int g (int *p, char *q) {
  ++*p;
  ++*q;
  return *p;
}

f:   movl    4(%esp), %ecx
     movl    8(%esp), %edx
     movl    (%ecx), %eax
     addl    $1, %eax
     movl    %eax, (%ecx)
     addl    $1, (%edx)
     ret
```

*p %ecx
*q %eax
p+1 %eax

```
g:   movl    4(%esp), %ecx
     movl    8(%esp), %edx
     movl    (%ecx), %eax
     addb    $1, (%edx)
     addl    $1, %eax
     movl    %eax, (%ecx)
     ret
```

*p %ecx
*q %edx
p %eax

There's a compiler bug: one of these functions is translated incorrectly, and the other one is OK. Identify the bug, and explain why the other function is translated correctly even though one might naively think that its translation has a similar bug.

6 (12 minutes). Suppose we have allocated memory locations 0xffff0000 through 0xffffffff for the stack, and we are worried that our x86 program might overflow the stack. We decide to institute the ironclad rule that if a function ever attempts to grow the stack past the allocated bounds, the function immediately stops what it's doing and returns 0, thus shrinking the stack. Explain the problems you see in implementing this rule. Don't worry about the effects of this rule on the user program; worry only about implementing the rule correctly.

7 (12 minutes). Give C source code that corresponds to the following x86-64 assembly language code. Explain briefly and at a high level what useful thing the function does.

```
sub:  movq   %rdi, %rdx      rdi = x
      subq   %rsi, %rdx      rsi = y
      xorq   %rdi, %rsi      rdx = temp
      xorq   %rdi, %rdx
      movq   %rdx, %rax      ((x-y)^x) & (y^x)
      andq   %rsi, %rax
      shrq   $63, %rax
      ret
```

```
bool sub(long x, long y){
    long temp = x-y;
    y ^= x;
    temp ^= x;
    return (temp & y >>63) & 0x01;
}
```

It returns true if x-y will overflow & false otherwise

8 (12 minutes). Match each of the following C source functions to each of the following assembly-language functions. A "match" means that the assembly-language code properly implements the C code. For example, if the C function 'f' is implemented by the assembly-language implementation 'B', write "f=B'.

```
int a(int x) { while (x & 1) x>>=1; return x; }
int b(int x) { while (x & 3) x>>=1; return x; }
int c(int x, int y)
  { return x / y - (x % y < 0); }
int d(int x, int y)
  { return x % y + (x % y < 0 ? y : 0); }
int e(unsigned x, unsigned y)
  { return (x + y < x) ^ ((int) y < 0); }
int f(int a, int b, int c) { return a ? b : c; }
int g(int a, int b, int c)
  { return a ? b + c : b & c; }
int h(unsigned x, unsigned y)
  { return x - x / y * y; }
int i(int x, int y) { return x - x / y * y; }
int j(int x) { return -~x; }
int k(int x) { return ~-x; }
int l(int x) { return x+~x; }
```

(continued on next page)

b=A ✓
a=D ✓
g=B ✓
f=C ✓
k=F ✓
j=I ✓
l=G ✓
e=J ✓
d=L ✓
h=H ✓
~~i=E~~
~~c=K~~

10

(continued from previous page)

```
A:      movl    4(%esp), %eax
        testb   $3, %al
        je      .L2
.L3:    sarl    %eax
        testb   $3, %al
        jne     .L3
.L2:    ret

B:      movl    8(%esp), %eax
        movl    12(%esp), %edx
        movl    %eax, %ecx
        orl     %edx, %ecx
        andl    %eax, %edx
        movl    4(%esp), %eax
        testl   %eax, %eax
        movl    %ecx, %eax
        cmove   %edx, %eax
        ret

C:      movl    4(%esp), %eax
        testl   %eax, %eax
        movl    12(%esp), %eax
        cmovne  8(%esp), %eax
        ret

D:      movl    4(%esp), %eax
        testb   $1, %al
        je      .L16
.L17:   sarl    %eax
        testb   $1, %al
        jne     .L17
.L16:   ret

E:      movl    4(%esp), %eax        [ x ] %eax
        cltd
        idivl   8(%esp)
        shrl    $31, %edx
        subl    %edx, %eax
        ret

F:      movl    4(%esp), %eax        [   ]
        subl    $1, %eax
        ret
```

$\sim x + 1 = -x$

$\sim x = -x - 1$

```
G:      movl    $-1, %eax
        ret

H:      movl    4(%esp), %eax        [ x ] %eax
        xorl    %edx, %edx
        divl    8(%esp)              [ 0x00000000 ] %
        movl    %edx, %eax
        ret

I:      movl    4(%esp), %eax
        addl    $1, %eax
        ret

J:      movl    8(%esp), %edx
        movl    %edx, %eax
        addl    4(%esp), %eax
        setc    %al
        shrl    $31, %edx
        xorl    %eax, %edx
        movzbl  %dl, %eax
        ret

K:      movl    4(%esp), %eax
        cltd
        idivl   8(%esp)
        movl    %edx, %eax
        ret

L:      movl    4(%esp), %eax        [ x ] %eax
        movl    8(%esp), %ecx        [ y ] %ecx
        cltd
        idivl   %ecx
        movl    $0, %eax
        testl   %edx, %edx
        cmovns  %eax, %ecx
        leal    (%edx,%ecx), %eax
        ret
```

9 (12 minutes). Consider the following program:

```
1  unsigned ack (unsigned m, unsigned n) {
2    if (m == 0)
3      return n + 1;
4    if (n == 0)
5      return ack (m - 1, 1);
6    return ack (m - 1,
7                ack (m,
8                     n - 1));
9  }
```

(continued in next column)

21/30

and the following assembly-language implementation as reported by objdump:

```
1   ack:
2                  pushl   %ebx   - callee save
3                  subl    $8, %esp
4   m %ebx         movl    16(%esp), %ebx  line 1   +4
5   n %eax         movl    20(%esp), %eax  line 1
6                  testl   %ebx, %ebx  line 2
7             +2   je      .L2  2,3 this is the if statement
8                  subl    $1, %ebx  5 & 6 both have m-1
9             +2   jmp     .L4  transition from line 2→4
10  .L7:
11                 testl   %ebx, %ebx → line 2
12                 movl    $1, %eax
13                 leal    -1(%ebx), %edx
14                 je      .L2  jump from line 2→3
15  .L6:
16                 movl    %edx, %ebx  line 5 & 6 both have m-1
17  .L4:
18                 testl   %eax, %eax  line 4
19  m+1 %edx       leal    1(%ebx), %edx
20                 je      .L7  jump line 5→2
21                 subl    $8, %esp   - decrement stack
22                 subl    $1, %eax  line 8
23                 pushl   %eax  ] caller save
24                 pushl   %edx  ]
25                 call    ack  line 7
26                 addl    $16, %esp  allocate stack frame
27                 testl   %ebx, %ebx  line 2
28  m-1 %edx       leal    -1(%ebx), %edx  line 5 & 6 both have m-1
29                 jne     .L6  jump 2→5 or 6
30  .L2:
31                 addl    $8, %esp  - decrement stack
32                 addl    $1, %eax  - line 3
33            -4   popl    %ebx   - callee restore
34                 ret   line 3
```

For each instruction in the implementation, identify the corresponding source-code line number. If an instruction corresponds to two or more source-code line numbers, write them all down and explain.