CSCI 356 Fall 2017 : Practice Exam I

DO NOT OPEN EXAM PACKET UNTIL INSTRUCTED TO DO SO

YOU MAY FILL IN INFORMATION ON THE FRONT NOW

PLEASE TURN OFF ALL ELECTRONIC DEVICES

ID#:	
Name:	

- This exam is closed book. You are allowed one (1) 8.5” x 11” handwritten note sheet
- You will have eighty (80) minutes to complete this exam.
- Answer the questions only in the spaces provided on the question sheets.
- If you give multiple solutions to a problem without indicating which one you want graded, the grader may select one to grade.
- Your answers do not need to be complete, grammatically correct sentences.

Problem	Points	Possible
1		8
2		4
3		4
4		6
5		4
6		8
7		6
8		7
Total		47

1. Solve the following project one style problem.

```
/*
 * isAsciiDigit - return 1 if 0x30 <= x <= 0x39 (ASCII codes for characters '0' to '9')
 *   Example: isAsciiDigit(0x35) = 1.
 *            isAsciiDigit(0x3a) = 0.
 *            isAsciiDigit(0x05) = 0.
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 15
 */

int isAsciiDigit(int x) {
        int neg_0x30 = ~(0x30) + 1;
        int x_minus_0x30 = x + neg_0x30;
        int neg_0x3a = ~(0x3a) + 1;
        int x_minus_0x3a = x + neg_0x3a;
        int lower_bound_mask = !(x_minus_0x30 >> 31); //0 if x >= 0x30, 1 if x < 0x30
        int upper_bound_mask = !!(x_minus_0x3a >> 31); //0 if x > 0x3a, 1 if x <= 0x39
        return lower_bound_mask & upper_bound_mask; //TOTAL OPS: 12
}
```

2. Write the following base-10 integers in **eight-bit two's complement**. Express your answer in both binary and hex (base-16).

   a. 53

      53 = 32 + 16 + 4 + 1
      53 = 0011 0101
      0011 -> 3 0101 -> 5 53 = 0x35

   b. -75
      -75 = -128 + 32 + 16 + 4 + 1
      -75 = 1011 0101
      1011 -> 11 -->b 0101 -> 5 -75 = 0xb5

3. Interpret the following as hex representations of **two's complement integers (eight bits each)**. Write them both in binary and in base-10.

   a. 0xCF
      c → 12 → 1100 f → 15 → 1111 0xCF = 1100 1111
      0xCF = -128 + 64 + 8 + 4 + 2 + 1 = -49

   b. 0x49
      4 → 0100 9 → 1001 0x49 = 0100 1001
      0x49 = 64 + 8 + 1 = 73

4. Consider the **eight-bit floating point format**. In eight-bit floating point, there is one sign bit, three exponent bits, and four fractional bits. The exponent bias is 3.

   a. What number is 0110 1100 in base 10?

      e = 110 = 6
      f = 0.1100 = 0.75
      E = e - bias = 6 - 3 = 3
      M = 1 + f = 1 + 0.75 = 1.75
      sign bit = 0 → positive
      $1.75 \times 2^3 = 1.1100 \times 2^3 = 1110.0 = 14$

   b. How would 3.3125 (= 3 + 5/16) be represented in eight-bit floating point?

      3.3125 = 11.0101 = 1.10101×2^1
      M = 1.10101 → f = 0.**10101**
      E = 1 → e = 1 + 3 = 4 = 100
      S = positive = 0
      3.3125 = **0100 1010** 1

5. Give a value that makes each following expressions false, and explain why it makes the expression false. If there is no value for x and y that would make the expression false, indicate that. In each case, x and y are of type int.

   a. ((x^y) < 0)

      If x is 0x0, y is 0x0

   b. ((x >> 31) + 1) >= 0

      This is always true. There's two cases: x >= 0 or x < 0. If x >= 0, x >> 31 is 0x0. 0x0 + 1 = 0x00000001. That's greater than 0. If x < 0, x >> 31 is 0x11111111. 0xffffffff + 1 = 0. That's equal to 0.

6. I have a C function with the following signature:
   `int practice_exam_problem(int a, int b);`

Here is the assembly code for it:

```
 <+0>:cmp     %esi,%edi
<+2>:  jle    0x4005be <practice_exam_problem+12>
<+4>:  lea    0x5(%rsi,%rsi,1),%eax
<+8>:  cmp    %eax,%edi
<+10>: je     0x4005d4 <practice_exam_problem+34>
<+12>:cmp     %esi,%edi
<+14>:jge     0x4005ca <practice_exam_problem+24>
<+16>:lea     0x4(%rdi,%rdi,2),%eax
<+20>: cmp     %eax,%esi
<+22>: je     0x4005da <practice_exam_problem+40>
<+24>: cmp     %esi,%edi
<+26>: jne    0x4005e0 <practice_exam_problem+46>
<+28>: mov     $0x4,%eax
<+33>: retq
<+34>: mov     $0x3,%eax
<+39>: retq
<+40>: mov     $0xa,%eax
<+45>: retq
<+46>: mov     $0x2,%eax
<+51>:retq
```

a. Give a value for parameters to make it return 2.

   To return 2, we jump from <+26>, which triggers if %esi != %edi. A good way to get to <+26> is from <+14>, which jumps to <+24> iff %edi >= %esi. But we want to NOT trigger the jump on <+10> checking if %edi == %eax, where %eax is 2 * %rsi + 5.

   Careful not to trigger the jumps on <+10> and <+2>!
   For the path to work, these conditions must hold:
   a < b
   b != 3a + 4

   OR

   a > b
   a != 2b + 5

b. Give a value for parameters to make it return 3.

The line that returns 3 is on <+34>. We jump to that from <+10>. That only happens when %eax == %edi. %eax is 2 * %rsi + 0x5. So this returns three when 2* %rsi + 0x5 == %edi.

So a == 2b + 5.

c. Give a value for parameters to make it return 4.

Nothing jumps to <+28>, but <+14> jumps to <+24>, which is right before it. To get there, we need to jump from <+2>, the compare instruction behind <+14> So to make the jump to <+14> b <= a. To make the jump to <+24> a <= b. We want to avoid the jump to <+46>, so we want a == b. For all three conditions to hold true, we can just set a == b.

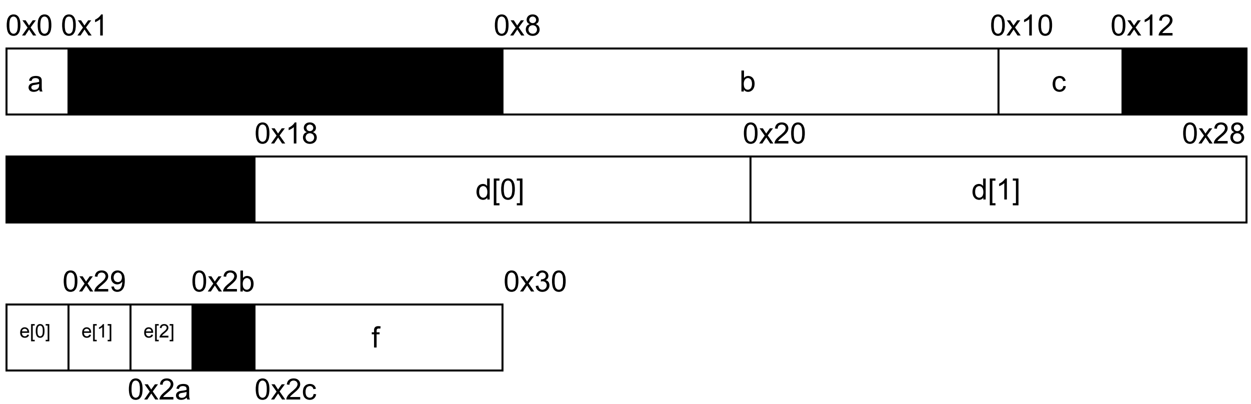
d. Give a value for parameters to make it return 10.

In order to return 10, we need to take the jump at <+22>, which happens when %eax == %esi. Before the comparison, %eax is set to 3 * %rdi + 4, which is 3a + 4; so we want b == 3a + 4. We also want %edi < %esi so we don't jump at <+14>. Finally, we load 2 * %rsi + 5 into %eax and compare that to %edi; we don't want to take the jump, so we need a != 2b + 5.

Final restrictions:
b == 3a + 4
a != 2b + 5
a < b

7. Consider the following struct on an x86-64 Linux machine:

```
struct my_struct {
      char a;
      long b;
      short c;
      float *d[2];
      unsigned char e[3];
      float f;
};
```



a. How many bytes will the struct occupy if our compiler optimizes for access time?

48 bytes

b. How many bytes will the struct occupy if our compiler optimizes for space?

34 bytes

8. Draw the stack frames of test and getbuf, given that the Instruction Pointer is currently at 0x004017c7 and the stack pointer is at 0x5561dcac at the start of test. Indicate where the stack pointer is and the addresses and the content of the stack frames (variable names are ok).

C code:

```
void test() {
    int val;
    val = getbuf();
    printf("No exploit. Getbuf returned 0x%x\n", val);
}
```

```
unsigned getbuf() {
    char buf[BUFFER_SIZE];
    gets(buf);
    return 1;
}
```

Assembly Code:

test:		
0x00401984	sub	$0x8,%rsp
0x00401988	mov	$0x0,%eax
0x0040198d	callq	4017c3<getbuf>
0x00401992	mov	%eax,%edx
0x00401994	mov	$0x4031d8,%esi
0x00401999	mov	$0x1,%edi
0x0040199e	mov	$0x0,%eax
0x004019a3	callq	400e00<__printf_chk@plt>
0x004019a8	add	$0x8,%rsp
0x004019ac	retq	
getbuf:		
0x004017c3	sub	$0x28,%rsp
0x004017c7	mov	%rsp,%rdi
0x004017ca	callq	401a4d <Gets>
0x004017cf	mov	$0x1,%eax
0x004017d4	add	$0x28,%rsp
0x004017d8	retq	

Stack

	Address	Contents
test stack frame	0x5561dcac	
	0x5561dca4	
	0x5561dc9c	0x401992
getbuf stack frame	0x5561dc94	
	0x5561dc8c	
	0x5561dc84	
	0x5561dc7c	
	0x5561dc74	

Instruction Pointer	0x004017c7
Stack Pointer	0x5561dc74

```c
/*
 * anyOddBit - return 1 if any odd-numbered bit in word set to 1
 *   Examples anyOddBit(0x5) = 0, anyOddBit(0x7) = 1
 *   Legal ops: ! ~ & ^ | + << >>
 *   Max ops: 12
 *   Rating: 2
 */
int anyOddBit(int x)  {
   int m8 = 0xAA;
   int m16 = m8 | m8 << 8;
   int m32 = m16 | m16 <<16;
   int oddx = x & m32;
   return !!oddx;

}
```